AUTONOMOUS DIAL - A - RIDE TRANSIT SYSTEM : IMPROVEMENT METHODS

by

Deepak Mittal

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1996

AUTONOMOUS DIAL-A-RIDE TRANSIT SYSTEM : IMPROVEMENT METHODS

A Thesis submitted

In Partial Fulfillment of the Requirements for the degree of

MASTER OF TECHNOLOGY

by

DEEPAK MITTAL

to the

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

APRIL, 1996.

CERTIFICATE

This is to certify that the work contained in this thesis entitled "Autonomous Dial-a-Ride Transit System : Improvement Methods" by Deepak Mittal (Roll No. 9411408) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Ravindra K. Ahuja)
Professor
Industrial and Management Department
Indian Institute of Technology
Kanpur - 208016

April, 1996.

ACKNOWLEDGEMENTS

First and Foremost I express my heartfelt thanks to Dr. R. K. Ahuja. Despite his demanding schedule, he made himself available whenever I required his guidance.

I am also thankful to my colleagues Kansal, Puneet, Kapil, Jha, Shobhit, Parvesh, Rishi, Madhav and Gopal for their invaluable support in carrying out this work. I am also grateful to all members of IME family for their valuable suggestions, direct or indirect, during the course of my thesis.

April, 1996 Deepak Mittal

ABSTRACT

In this thesis, we have developed heuristic algorithms for autonomous dial-a-ride transit (ADART) system. This problem is concerned with developing a set of routes for a fleet of vehicles serving customers who have to be picked up from specified origins and be delivered to specified destinations. The algorithms used for obtaining initial solutions have been adapted from the doctoral thesis of J. J. Jaw. There are two versions of the dial-a-ride problem : advance request version, and immediate request version.

For advance request version, we developed four heuristic approaches for improving the initial solution. In the first approach, we are swapping the nodes of a route generated by a construction algorithm. In the second approach, we are enumerating many routes for a vehicle and selecting the best route among them. In the third approach, we are globally interchanging the customers among existing routes. In the fourth approach, we are running the construction algorithm twice to allow the customers to have more opportunities for insertions.

For immediate request version, we are applying swapping strategy after each insertion of a customer to improve the solution. We have also incorporated the enumeration approach for handling the immediate requests.

We have developed algorithm to handle cancellation of requests. Further, we have developed algorithm for private taxi, so that we can compare the ADART service with private taxi.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION TO AUTONOMOUS DIAL-A-RIDE TRANSIT SYSTEM

Autonomous dial-a-ride Transit (ADART) system is a mass transit system which would serve areas of high travel demand. It is a modernized version of many-to-few dial-a-ride transportation system. It employs fully automated order-entry and dispatching systems that reside on board the vehicle. An ADART vehicle fleet efficiently serves travel demand in a large geographical area without need of centralized supervision.

Each vehicle's on-board computer receives a customer's trip request, inserts this request into the vehicle's schedule, and plans an optimal route to accomplish the schedule. The computer controls the vehicle's route by continuously informing the driver of the next required change of direction. Furthermore, if advantageous to do so, the computer may pass responsibility for a trip request off to another vehicle-computer better positioned to provide the service.

Due to its distributed command-and-control, ADART promises to provide better service than conventional dial-a-ride and at lower operating cost. It increases driver productivity and reduces total manpower. It can readily increase or reduce its fleet size at a fixed operating cost per vehicle. Redeploying vehicles from one service area to another is almost free.

Furthermore, in contrast with conventional dial-a-ride, the greater the demand, the better the ADART system performs. The result could be a transportation service for low density areas that is attractive to customer and supplier alike.

The concept of Autonomous dial-a-ride service is simple, the difficult part is to operate such a system efficiently. This thesis is devoted to the live component of a set of tools that can help to operate such vehicles in an efficient manner. The focus of our work is on designing computerized algorithms which will decide the routes of vehicles. In the following section, we will discuss various types of autonomous dial-a-ride problems that might arise in this context. In Section 1.2 we explain the purpose of this thesis and present an outline of the work.

1.2 PURPOSE OF THIS THESIS

Most conventional dial-a-ride system need centralized supervision. Correspondingly, their routing and scheduling algorithms are centralized algorithms. In ADART, the use of centralized supervision is minimized and decision making is decentralized. Accordingly, we propose to develop a new set of decentralized algorithms to accomplish the following tasks :

1. Schedule advance trip requests
2. Modify schedules based on new trip requests and cancellations.
3. Continuously improve the routes of vehicles.

Chapter 2 of this thesis gives an overview of past work on various versions of the dial-a-ride problems. Chapter 3 describes the Autonomous Dial-A-Ride Transit System, and ADARTW algorithm. Chapter 4 introduces different algorithms for scheduling advance trip requests, modifying schedules based on new trip requests or cancellations, and for continuously improving the routes of vehicles, and computational results of different algorithms. Chapter 6 suggests strategies which may improve the algorithms and concludes this work by suggesting possible areas of future research.

CHAPTER 2: REVIEW OF ALGORITHMS FOR DIAL-A-RIDE PROBLEMS

2.1 INTRODUCTION

This chapter reviews earlier work that has investigated the dial-a-ride problem. The discussion consists of two parts. In Section 2.2, single vehicle algorithms are discussed. In Section 2.3 multi-vehicle algorithms are discussed.

2.2 SINGLE VEHICLE ALGORITHMS

Algorithms for the advance-request dial-a-ride problem can be divided into two categories: single-vehicle or multi-vehicle algorithms. Since the single-vehicle problem is a subproblem of the multi-vehicle case, its algorithm are mostly used as "subroutines" in the multi-vehicle problem. The single-vehicle advance-request dial-a-ride problem can be defined as follows: Given N pairs of pick-up and delivery points, find the shortest route covering all the points with a customer's pick-up point visited first before visiting his delivery point. This problem is similar to the traveling salesman problem with precedence constraints. The objective function can vary from the basic shortest route length to include some measurement of customers' disutility.

Time constraints can also be added to the basic problem so that some quality of service can be guaranteed.

In [2,3], Psaraftis introduced an polynomial time heuristics for the single vehicle dial-a-ride problem. In [7], Sexton and Bodin introduce a heuristic algorithm for the single vehicle and multi-vehicle dial-a-ride problem. In [6], Solomon and others introduced an algorithm for route improvement procedures. Solomon used branch exchange solution improvement procedures for the vehicle routing and scheduling problems with time window constraints. In [1], George Kotoraclvis and Jonathan F. Bard introduced greedy randomized adaptive search procedures to solve multi vehicle advance trip request problem. We now discuss these approaches in detail.

(a) Two polynomial-time heuristics

Psaraftis [2,3] has developed two polynomial-time heuristics for the single vehicle dial-a-ride problem with the objective of minimizing route length. Time constraints and customer disutility are not considered in this case. The first heuristic is rather simple: It is based on the Minimum Spanning Tree (MST) that is defined on the N origins and the N destinations of the problem. From the MST in question, an initial traveling salesman tour T_0 through the above 2N points can be constructed. Choose any customer origin on T_0 as the first pick-up point P_1 on the dial-a-ride route. From this point, move on T_0 clockwise until all points are visited and then return to A (the initial vehicle location). While doing this, do not visit any point that has bee previously visited, or any destination whose origin has not been previously visited. Call this dial-a-ride tour T_1.

The procedure described above is a generic version of the heuristic. Several [illegible] are possible: (a) Instead of moving clockwise on T_0, we can

Time constraints can also be added to the basic problem so that some quality of service can be guaranteed.

In [2,3], Psaraftis introduced an polynomial time heuristics for the single vehicle dial-a-ride problem. In [7], Sexton and Bodin introduce a heuristic algorithm for the single vehicle and multi-vehicle dial-a-ride problem. In [6], Solomon and others introduced an algorithm for route improvement procedures. Solomon used branch exchange solution improvement procedures for the vehicle routing and scheduling problems with time window constraints. In [1], George Kotoraclvis and Jonathan F Bard introduced greedy randomized adaptive search procedures to solve multi vehicle advance trip request problem We now discuss these approaches in detail.

(a) Two polynomial-time heuristics

Psaraftis [2,3] has developed two polynomial-time heuristics for the single vehicle dial-a-ride problem with the objective of minimizing route length. Time constraints and customer disutility are not considered in this case. The first heuristic is rather simple: It is based on the Minimum Spanning Tree (MST) that is defined on the N origins and the N destinations of the problem. From the MST in question, an initial traveling salesman tour T_0 through the above 2N points can be constructed. Choose any customer origin on T_0 as the first pick-up point P_1 on the dial-a-ride route. From this point, move on T_0 clockwise until all points are visited and then return to A (the initial vehicle location). While doing this, do not visit any point that has bee previously visited, or any destination whose origin has not been previously visited. Call this dial-a-ride tour T_1.

The procedure described above is a generic version of the heuristic. Several

choose to move counterclockwise and compare with the previous result (b) Each time choose a different customer origin as P_1. (c) Improve upon T_1 by performing a sequence of local interchanges (see details below).

The computational complexity of this heuristic in $O(N^2)$. A pathological case is constructed which shows that the relative error of the heuristic can potentially go as high as 300%. However, the average performance of the heuristic is projected to be 13% off the optimum by computing simulation results with the asymptotic value

In the optional step (c) of the MST-based heuristic, it is indicated that T_1 can be improved by applying a sequence of local interchanges As the second polynomial-time heuristic, Psaraftis [2,3] has designed a k-interchange procedure for the TSP. As in the TSP, a k-interchange in the dial-a-ride problem is a substitution of k of the link of a dial-a-ride tour with k other links. A dial-a-ride tour is said to be k-optimal (or k-opt) if it is impossible to obtain another dial-a-ride tour of shorter length by replacing any K of its links by any other set of K links

In [2,3], Psaraftis was able to develop a method that finds the best k-interchange that can be produced from an initial feasible dial-a-ride tour in $O(N^k)$ time for the TSP. Psaraftis has described in detail the k-interchange procedure for the cases of k = 2 and k = 3. Computational experimentation shows that the 3-opt procedures outperform the corresponding 2-opt algorithms by producing tours that are about 30% shorter on the average if the initial tour is random and about 6% shorter on the average if the initial tour is MST-generated.

(b) Bender's Decomposition

In [7], Sexton and Bodin introduce a heuristic algorithm aimed at minimizing total inconvenience for all N customers. The total inconvenience D_i of customer i is defined to be a weighted sum of his excess ride time, ERT_i, and his delivery time deviation, DV_i (all customers specify a desired delivery time)

$$D_i = A \cdot ERT_i + B \cdot DV_i \qquad A, B \text{ are given constants.}$$

The problem is to find a schedule that minimizes $\sum_{i=1}^{N} D_i$

Constraints of the problem include that vehicles have limited capacity and customers cannot be delivered later than their desired delivery times.

The algorithm can be outlined as follows :

Step 1 Use a space-time heuristic to form an initial feasible route.

Step 2 Find the optimal schedule for the given route by converting the problem to a maximum profit network flow problem which can be solved without substantial computational effort.

Step 3 Improve upon the solution at hand by changing the objective function coefficients of the routing problem and see whether the new route and schedule produced yields better results in terms of total customer inconvenience.

This algorithm was coded in FORTRAN and implemented on a UIVAC 1100/70 computer. Test data were obtained from operating sites with problem size ranging from 7 - 20 customers per vehicle. Computation time averaged

roughly 18 seconds of CPU time per vehicle. This algorithm is later used as a core component in a multi-vehicle algorithm by the same author.

(c) Vehicle routing and scheduling problem with time window constraints by Solomon and others [6]

They extended branch exchange solution improvement procedure to vehicle routing and scheduling problems with time window constraints. They discussed the branch exchanges heuristics in two parts, which are as follows:

- Exchanges involving two branches, 2-opt
- Exchanges involving three branches, 3-opt

The notations used in this section are as follows :

e(i) = the earliest arrival time at customer i

f(i) = the latest arrival time at customer i

s(i) = the service time for customer i

t(i) = the time at which service begins for customer i

a(i, j) = the travel time from i to j

c(i, j) = the cost of traversing arc (i, j)

Exchanges involving two branches, 2-opt

In this study, the service time for each customer was considered to include any minimum dwell time. The cost of traveling time from i to j, c(i, j), was assumed to be equal to a function of travel time and distance. In the VRSPW, each customer was assumed to have one or more time windows of the form:

The time at which service begins at customer i, t(i), was defined as

$$t(1) = \max\{e(i), t(j) + s(j) + t(j) + a(j, 1)\},$$

where j is the immediate predecessor of 1 and where the actual time at node i may be determined as

$$t(j) + s(j) + a(j, 1)$$

If a vehicle arrives at a customer before the time window opens, that is before e(i), then the vehicle must wait The wait at node i, w(1), is defined as .

$$w(1) = \max\{0, e(1) - (t(j) + s(j) + a(j, 1))\}$$

An example VRSPTW vehicle route is presented in Figure 2.1. Consider the 2-Interchange depicted in Figure 2.2. The proposed change would substitute arc (1, j) and (1+1, j+1) for arcs (1, i+1) and (j, j+1) respectively. Using total route distance as the performance measure, this 2-Interchange would result in a local route improvement if and only if :

$$c(i, j) + c(i+1, j+1) < c(i, i+1) + c(j, j+1).$$

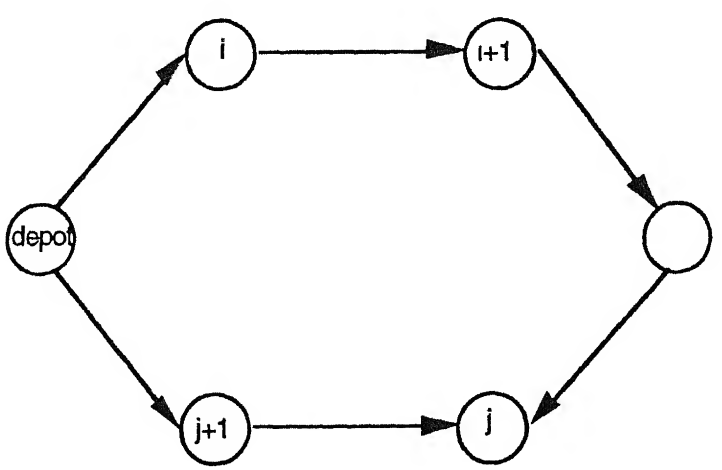


Figure 2.1

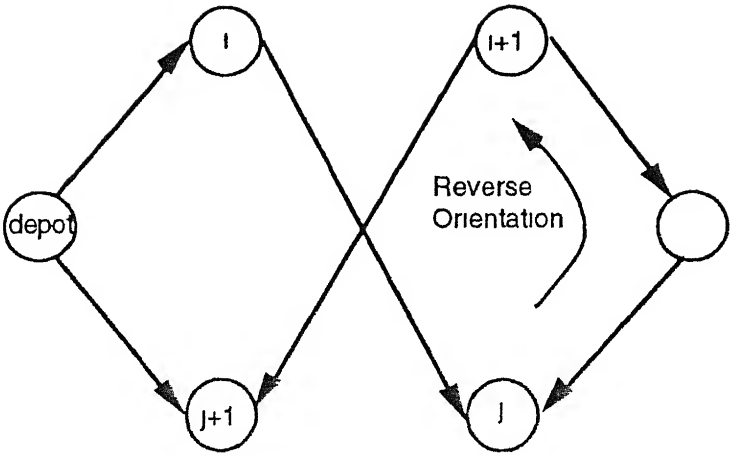


Figure 2.2

Additionally, the time window constraints of the customers affected by the exchange would also need to be satisfied This will include all customers from i+1 to the end of the route, but particularly those customers on the route between j and i+1 since this section of the route will now be traversed to the reverse direction Since the number of customers that may be contained to the section of the route between customers j and i+1 may vary, the computational efforts required to check the feasibility of each customer's time window is of $O(N_k)$, where N_k is the number of customers on route k This additional level of complexity in the 2-opt procedure would result in O(N**3) computation for the VRSPTW as opposed to the O(N**2) computations required in the standard TSP.

It is easy to determine whether it is necessary for customer i to precede customer j. For any vehicle route, an examination of all such precedence relationships may be performed in O(N**2) time. The resultant matrix may then be defined as:

$$VP(i, j) = \begin{cases} +1 \text{ if customer i must precede customer j;} \\ 0 \text{ if no precedence relationship exists;} \\ -1 \text{ if customer j must precede customer i.} \end{cases}$$

From the vehicle precedence matrix it is now possible to define a value for each customer on the route which reflects the precedence dependence at that point with regard to customer to be visited later on the route. For each customer i on the route, we define a node precedence value, NP(i), equal to the number of the first customer beyond customer i+1 that also has a predecessor which occurs after customer i on the route. They define.

NP(i) = the smallest value of k, k > i+1, such that

VP(j, k) = +1, j ≥ i+1. If no such k exists,

then set NP(i) = N_k+1.

The feasibility condition may be stated :

Condition: A necessary condition for the feasibility of the 2-exchange of arcs, (i, i+1) and (j, j+1) with arcs (i, j) and (i+1, j+1) is that j < NP(i).

The techniques discussed above for the 2-exchange heuristic may also be extended to the 3-exchange procedures

2.3 MULTI-VEHICLE ALGORITHMS

The multi-vehicle advance-request dial-a-ride problem has attracted much interest lately among several groups of researchers. We will examine other works on this problem in this section :

(a) Bender's Decomposition

Bodin and Sexton proposed a heuristic algorithm which employs the single vehicle dial-a-ride algorithm described in Section 2 2 as the core component for solving the multi-vehicle case. It first partitions customers into several clusters Within each cluster, the single-vehicle algorithm is used to develop the vehicle route and schedule which minimize customer inconvenience. After initial schedules are obtained for each vehicle, a route improvement procedure, the "swapper" algorithm, is used to swap customers one at a time between vehicles so that the total customer inconvenience is reduced. This improvement procedure is repeated until no further improvement can be found.

This algorithm was applied to one small data set consisting of about 85 afternoon customers of the dial-a-ride system in Baltimore, MD It took about 2-3 minutes CPU time on UNIVAC 1108 to complete a single run Results compared favorably with the manual solution

(b) A GRASP for the Vehicle Routing Problem with Time Windows by George [1]

In [1], George Kotoraclvis and Jonathan F. Bard introduce a heuristic algorithm aimed at finding the minimum number of vehicles required to visit a set of nodes subject to time windows constraints. Their secondary objective was to minimize the total distance traveled Their assumptions are as follows :

- Each node requires a predetermined amount of service in the form of pickups and deliveries.
- The fleet is homogenous and if located at common depot.

- Vehicle capacity is finite and spit service is not permitted.

A greedy randomized adaptive search procedures (GRASP) is used to obtain feasible solutions In their heuristic, construction of feasible solutions is performed by first initializing a number of routes, r, and then iteratively assigning one customer at a time in accordance with a adaptive set of rules. If at some step no unassigned customer can be inserted in an existing route a new route is created. The number of initial route can be anywhere from 1 to n The number of initial routes, r is set to a predetermined lower bound. This is preferable since you will never introduce more routes than necessary, thus reducing the effort required in the improvement phase to remove simpler routes.

To initialize the routes, seed customers have to be selected and assigned Solomon proposes two criteria for doing this

- maximum distance from the depot or in earliest deadline. They selected the seed customers that are either the most geographically dispensed
- the most time constrained.

The additional effort required in their computation is justified by preliminary results which showed that seed selection is a major determinant in the effectiveness of the GRASP.

An algorithm to eliminate the routes was also designed. The routes having fewer customers examined first. In order to eliminate route ρ all of its customers must be assigned to other routes. Two cases were considered. In the first case, attempt was to move K from its current route ρ to another route ρ'. If more than one alternative exists two possible ways of selecting K's new

route are considered. The first and the simplest is to select the first route found, the second is to select the route that provides the greatest savings in distance Although both alternatives have the same worst case complexity, in practice, the former is measurably faster

CHAPTER 3 DESCRIPTION OF AUTONOMOUS DIAL-A-RIDE TRANSIT SYSTEM

3.1 IDEAS BEHIND ADART

For twenty five years, dial-a-ride transit (DART) has failed to fulfill its promise. In actual DART deployments, either the service level was too low or the costs too high. Small systems lacked the coverage to achieve high demand. Large systems failed because operating costs climbed faster than revenue. In a word, DART could not cope with success.

This approach promises improved service and reductions in operating costs. Furthermore, these benefits accrue whatever the size of the DART system, becoming more significant as the system gets larger. Section 3.2 describes the benefits of ADART. Section 3.3 describes the ADART features. Section 3 4 describes about the types of ADART service. Section 3.5 describes multi-vehicle service areas. Section 3.6 describes fully automated dispatching system of ADART service. Section 3.7 describes about the routing and scheduling system of ADART service. Section 3.8 gives the introduction of ADARTW algorithm. In Section 3.9, we will describe the structure of ADARTW and the heuristic techniques that J. J. Jaw [4] uses. These are to a large extent dictated by the operating scenario within which the algorithm has been conceived. this Section also defines the mathematical notations used in the subsequent sections. Section 3.10 presents an overview of ADARTW. Section 3.11 describes the search for feasible insertions of customers into vehicle work-

schedules. Finally, Section 3.12 describes the optimization procedure used to assign customers to vehicles and to fix pick-up and delivery times. Section 3.13 describes about the generation of simulated data

3.2 ADART BENEFITS

The marketing, operational, accounting and technical benefits of ADART are simple:

The **Marketing Component** targets recurring trips, which account for over 50 percent of urban travel. These include trips to attraction centers for work, shopping, and personal purposes. ADART targets these recurring trips because they form a lucrative market. They are serviceable with a many-to-few operation and provide "repeat business," the life blood of an enterprise profiting on high volume.

The **Operational Component** eliminates communication with telephone operators to arrange travel, by providing a customer interface similar to *automatic teller machines* (ATM) and banking on phone. ADART passengers never converse with telephone operators, only with computers *on board the vehicle.*

The **Accounting Component** uses bank credit cards and electronic funds transfer for processing accounts receivable. This eliminates traditional fare collection.

The **Technical Component** eliminates the centralized dispatching function with a fully automated system, which assigns trips to vehicles, devises

itineraries, and plans routes *without any human intervention;* and has its dispatching hardware and software *on board the vehicle*

Each vehicle's computer communicates with its vehicle's driver and other computers Having no central control, an ADART fleet behaves like a swarm of ants doing their work with no one in charge [1]

3.3 ADART FEATURES

From the customer's point of view, ADART costs more than a bus but provides a better service It costs less than a taxi but provides a lesser service. It ignores the off-the-street occasional users ADART has four salient features or advantages subscription use, many-to-few service, convenience, and reliability.

From the operator's point of view, ADART's important features include cashless/checkless revenue collection, fully automated dispatching, on-board information, command and control systems, high driver productivity, low operating overhead, and easy adaptability to demand.

[1]This is not to suggest that an ADART operation lacks central management Vehicle and data maintenance, technical trouble-shooting, service quality control, pricing, hiring, fixing, automatic call distribution, and accounting are a few examples of functions best served by ADART's central management The paper *does* say that ADART has negligible centralized intervention (mechanical or

3.4 TYPES OF DART SERVICE

Conventional DART service falls into one of the three categories: many-to-one, many-to-few, and many-to-many. All three imply *two-way* service

Many-to-One (MTO)

The simplest service to provide, it transports passengers to or from many locations (e g , residences) to one location (e g , shopping complex) The user of MTO service has only to provide one trip-end location from the "many," the other end of his trip is always the "one "

Many-to-Few (MTF)

In contrast to MTO service, MTF serves more than one attraction center, and is harder to provide.

Every city has several market segments of its traveling public that MTF service can target Since many trips go from home to some attraction center, MTF coverage is high enough to attract sufficient ridership, which maintain vehicle load factors that sustain service at moderate cost.[2] This makes MTF the most cost-effective form of DART.

Many-to-Many (MTM)

In fact, MTM is a more general case of MTF. In this both trip ends are scattered.

---

[2]Any ADART deployment would first require substantiation of these claims with site-specific demand

3.5 MULTI-VEHICLE SERVICE AREAS

The most productive dial-a-ride systems allow more than one vehicle to pick-up passengers within the same service area. Attempting to simplify dispatching, the most primitive demand-responsive systems partition a service area into non-overlapping sub-areas, and assign sub-area coverage exclusively to a single vehicle.

- **Single Vehicle Coverage.** Given that service reliability is a feature that customers value most, it is dangerous to assign exclusive coverage of a geographic sub-area to one vehicle.

- **Multi-Vehicle Coverage.** By assigning multiple vehicles to the same service area, DART can assign a trip to the vehicle that most efficiently serves it This assignment depends on the present and planned locations of all vehicles.

3.6 FULLY AUTOMATED DISPATCHING

A fully automated dispatching (FAD) system is one that can field a customer's request for service, schedule a vehicle to provide that service, and optimally route the vehicle *without any human interaction.* That is, the customer is the only human involved in the entire process of requesting a ride, scheduling arrivals and routing the vehicle. There is no "telephone center" to receive calls nor any "centralized dispatch" to assign trips to vehicles. The

driver merely obeys the driving directions he receives from his vehicle's computer.

The vehicle's on-board computer receives a customer request, inserts this request into the vehicle's schedule, and plans an optimal route to accomplish the schedule Furthermore, if beneficial to do so, the computer may pass the request off to another DART vehicle, without the drivers knowing it The DART vehicle's computer controls the vehicle's route by continuously informing the driver of the next required change of direction.

3.7 ROUTING AND SCHEDULING

Using every free computer cycle, the routing and scheduling system (RSS) continually works to improve the planned route of the vehicle to fulfill its outstanding trips. The RSS's goal is simply to devise an itinerary for the vehicle to pick-up and deliver would-be passengers according to their origin/destination and time-window requirements at minimum cost.

After composing an excellent route for its vehicle, the RSS might receive a new trip, whose inclusion destroys the viability of the planned route. Another complication is that the computer has limited time to find a feasible solution.

Trip-Vehicle Assignment

Whenever a user calls the ADART phone number, the vehicle's computer receives the call, prompts trip data from the customer. This vehicle's computer then initiates an "auction" for the trip.

This auction begins with the initiating vehicle/computer (VC) estimating the "marginal cost" (e g , expected miles/trip) of including that trip into its vehicle's schedule The computer figures this cost by inserting the trip into its vehicle's planned route in a way that assures the vehicle meeting all its scheduled arrivals.

The initial VC then broadcasts this cost and the trip's requirements to every vehicle serving the same sub-area. Each of these latter VCs makes its own estimate of including the trip in its own itinerary Any VC whose estimate is lower than the initial VC's responds with its own cost. The others remain silent

The initial VC then informs the VC with the lowest cost of its responsibility for the trip Thus responsible VC enters the trip into its scheduling system, which grinds away toward an optimal route to serve all its outstanding trips All other VCs ignore the trip.

In technical parlance, the vehicles' computers collectively assign the new trip to a "cluster" belonging to the responsible vehicle, thus leaving each vehicle's computer to solve only one small traveling salesman problem (TSP). Furthermore, all the vehicles' computers work on their particular TSP in parallel. This decomposes an otherwise huge, daunting problem into several easier small problems - all solved simultaneously - and enables an ADART operation to keep up with even the largest demand surges.

Note that no human participates in trip - vehicle assignment, routing or scheduling - not the driver, not the customer, and certainly not any dispatcher.

Trip-Vehicle Reassignment

A further important application of this auction algorithm occurs when a vehicle's RSS notes that its vehicle's projected productivity has dropped below some critical threshold. Here, a "problem trip" is uncovered by excluding each trip from the schedule and learning the maximum cost saving. The trip that saves the most is put up for auction, so that a better positioned vehicle might service it. This simple on-the-fly expedient improves productivity and reduces service delays

Vehicle Failure

The same algorithm supports fail-soft measures when a vehicle unpredictably goes out of service. Here, all the disabled vehicle's trips are auctioned off to the remaining vehicles.

Unserviceable Trips

In the rare event that an accepted trip request later becomes unserviceable, the responsible vehicle's computer calls a taxi for the trip.

3.8 An Algorithm for DARP with Strict Service Constraints

J. J. Jaw designed a heuristic algorithm, Advanced Dial-A-Ride With Time Windows (ADARTW), for the advance-request version of the multi-vehicle DARP with service-quality constraints that include time windows. Service quality constraints refer to guarantees that (i) each customer's ride time will not exceed a pre-specified maximum, and (ii) the time of pick-up or delivery

of a customer will not deviate from the most desired time by more than a pre-specified amount ("the time windows") The back bone of our approach is the model proposed by J. J. Jaw The later sections describes about J. J. Jaw algorithm.

3.9 OPERATING SCENARIO

We will first present the definitions and notations used in this chapter The various inputs to the system are :

DPT_i (DDT_i) · the desired pick-up (delivery) time of customer i

DRT_1 : the time it would take a vehicle to go directly from the origin to the destination of customer i, [DRT_1 = D(+i, -i)]

D(x, y) : the time it takes a vehicle to go from point x to point y (using fastest route)

WS_1 : the maximum acceptable deviation of customer i from his desired pick-up or delivery time ($DV_i \leq WS_1$)

Variables to the system are defined as follows :

N : the number of customers on the subscriber list

m : the number of available vehicles

EPT_i (EDT_i) : the earliest possible time at which the pick-up (delivery) of customer i can be made

LPT_i (LDT_i) : the latest possible time at which the pick-up (delivery) of customer i can be made

APT_i (ADT_i) : the time when the pick-up (delivery) of customer i will *actually* take place according to the schedule

ART_i · The actual ride-time of customer i, $ART_i = ADT_i - APT_i$

+i (-i) . the event "pick-up (deliver) customer i", the indication "+i" ("-i") is also used to denote the *point of origin* (*destination*) of customer i

DV_i · the deviation in the time schedule of customer i from his desired pick-up or delivery time [for DPT-specified customers $DV_i = APT_i - DPT_i$; for DDT-specified customers $DV_i = DDT_i - APT_i$]

d the number of stops (pick-ups and deliveries) in a schedule block

$SLACK_j$: the duration of vehicle slack time before schedule block j. If there are n schedule blocks, $SLACK_{n+1} = \infty$

The dial-a-ride system with which ADARTW is designed to operate is assumed to have a number of characteristics. The most important among those is that each of the system's customers is asked and willing to specify either a desired pick-up time (DPT) at his origin or a desired delivery time (DDT) at his destination, but not both. This implies that the customer is able to decide for himself whether he is constrained by a pick-up time or by a delivery time in his intended trip. For example, most individuals are constrained in the morning by a desired "delivery" time (e.g., the time one has to arrive at the work place) and adjust their "pick-up" time (e.g., the time when they leave their homes) accordingly. In a similar fashion, a dial-a-ride system's customer who specifies a desire to be delivered at a commuter train station (or an outpatient clinic) by time X, will be a "DDT-specified" customer.

In our system, this customer would rely on the operator of the system to tell his at what time he will be picked up from his origin so that he can be delivered at his destination by time X The reverse is, of course, true for DPT-specified customers (e g., "I can be picked up from the shopping mall at 2 P.M. for transportation to my house") Normally, one would expect a preponderance of DDT-specified customers in the morning and DPT-specified customers in the afternoon and evening.

A second and related assumption is that DPT-specified customers will be asked to give as their DPT, the earliest time at which they can be picked up. Similarly, DDT-specified customers will give the latest time at which they can be acceptably delivered at their destination as their DDT. This actually implies no loss of generality, but is particularly convenient for the algorithm, since the time-window during which a DPT- (DDT-) specified customer can be picked up (delivered) can be defined as beginning (ending) with the specified DPT (DDT).

In dial-a-ride system in question one would certainly expect a commitment of quality of service on the part of the system's operator. Consider, for instance, a DDT-specified customer who lives 15-minutes away (by car) from a community center and who desires to be at that center by 10 A.M. It would clearly be unreasonable to deliver that customer at the center at, say 8 A M. or to offer him a 90-minutes circuitous ride - picking up and/or delivering many other customers on the way. For these reasons it will be assumed that the system will operate under three types of service quality constraints :

(i) No DPT- (DDT-) specified customer will be picked up (delivered) earlier (later) than his DPT (DDT).

(ii) No customer's actual ride time will exceed a given maximum ride time for that customer - the maximum ride time being specified as a function of the direct origin-to-destination ride time for that customer.

(iii) The difference ("time deviation") between the actual pick-up (delivery) time and desired pick-up (delivery) time of a customer will not exceed a given maximum for DPT- (DDT-) specified customers.

The values of the maximum ride-time and of the maximum time-deviation can either be determined unilaterally by the system's operator and applied universally or, can be left open to negotiation between the operator and each individual customer. In the former case, an operator might advertise, for example, that a customer's ride time would "under no circumstances" exceed twice his direct ride time and that he would be delivered (picked-up) no earlier (later) than 20 minutes prior to (after) his desired delivery (pick-up) time

The problem

The version of DARP solved by ADARTW can now be summarized as follows : Given a subscription list of N customers, each specifying either a DPT_i or DDT_i (i = 1, 2, ..., N) and a fleet of m vehicles, find an effective allocation of customers among vehicles and an associated time schedule of pick-ups and deliveries such that :

1. For all customers i :

$$ADT_i - APT_i \leq MRT_i \qquad (3.1)$$

2. For DPT-specified customers :

$$DPT_i \leq APT_i \leq DPT_i + WS_i \qquad (3.2)$$

3. For DDT-specified customers :

$$DDT_i - WS_i \leq ADT_i \leq DDT_i \tag{3.3}$$

Several comments are in order concerning this formulation :

(a) We have not yet specified what is our measure of effectiveness (see Section 3 10 and 3.12).

(b) It may prove *infeasible* to serve some of the N customers with the given vehicle resources and service-quality constraints

(c) MRT_i, the maximum ride time for customer i, will normally be specified as a function of the direct ride time, DRT_i. In our work we have used :

$$MRT_i = A + B \,.\, DRT_i \tag{3.4}$$

where A and B are user-specified constraints (e.g., A = 5 minutes, B = 1.5). A reasonable alternative might be :

$$MRT_i = \begin{cases} DRT_i + A & \text{if } DRT_i \leq T_0 \\ B \,.\, DRT_i & \text{if } DRT_i > T_0 \end{cases}$$

where, again A, B and T_0 are operator-specified constraints (e.g., A = 10 minutes, T_0 = 20 minutes, B = 1.5) satisfying the relationship $T_0 + A = B \,.\, T_0$. Other functional forms can, of course, be used to specify MRT_i, if desired.

In conclusion, ADARTW is designed for use under the following scenario : The system's operator would advertise the dial-a-ride service and the service quality guarantees that will be offered. Customer will call and specify origin, destination and a desired DPT (= EPT) or DDT (= LDT). At the time, say K hour before the start of the service, initial routes will be generated by the computer on board the vehicle. These routes can be generated using

ADARTW algorithm based on complete information about all the customers that have requested service in advance

Before proceeding to the description of the algorithm a number of additional assumptions in our scenario will now be listed. While all these assumptions are natural ones, they are mentioned here because they create complications that some existing DARP algorithms cannot deal with :

a) The capacity of vehicles is assumed finite and it is not necessarily the same for all vehicles

b) Dwell times - the amount of time needed to pick up and deliver customers - can be non-zero and different customers are allowed to have different dwell times. It should be noted that non-zero dwell times can be handled by adding them to the distance matrix in a way consistent with the definitions of APT and ADT. We have taken dwell time to be 5 minutes for all customers by adding it to the time matrix.

c) A vehicle is not allowed to wait idly when it is carrying passengers. For example, it is not permissible in ADARTW to have a vehicle, with one or more passengers on board, arrive at the pick-up point of a DPT-specified customer i prior to DPT_i and then wait idly until time DPT_i to pick up i (remember that due to (3.2) customer i cannot be picked up earlier than DPT_i). Such idle waiting periods by non-empty vehicles are accepted by some vehicle-routing algorithms with time window constraints (see Sexton and Bodin [7]). It is felt, however, that, in particular, such idle waiting would not be tolerated by dial-a-ride customers and ADARTW has been designed accordingly. Actually this can be viewed as a fourth service-quality constraint, imposed in addition to (i) - (iii) above.

d) Each vehicle can have different origin and different destinations. Also each vehicle may be available in different time slots in a day.

Finally, it is useful to define *availability* periods, *active* periods and *slack* periods for vehicles. As show in Figure 3.1 for a particular vehicle j, a vehicle can be unavailable during periods of a day (usually due to driver constraints, labor union agreements or vehicle maintenance requirements) An available vehicle can be either in a slack period (i e., waiting idly) or it can be active (on the way to pick up it's first customer during an active period, transporting, picking up or delivering customers or returning to a depot). Note that because of assumption c) above, a vehicle cannot be in a slack period as long as it has even one customer on board

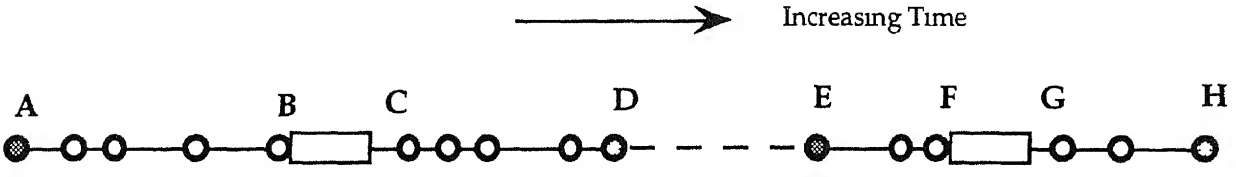


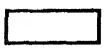
Figure 3.1

A typical schedule of a vehicle during service period AH


Denotes Origin of a Vehicle


Denotes Destination of a Vehicle


Denotes Pickup or Delivery of a Customer


Denotes Slack Time of the Vehicle during interval BC and FG

Denotes Unavailability of the Vehicle during interval DE

3.10 OVERVIEW OF THE ALGORITHM

ADARTW is a heuristic algorithm that processes ride requests sequentially, inserting one customer at a time into the work-schedule of some vehicle until all ride requests have been processed This section describes in qualitative terms how this procedure works.

Central to the process of assigning customers to vehicles are : a search for *feasible* insertions of customers into work-schedules; and a sequence of *optimization* steps designed to find the most desirable one among all the feasible insertions on each occasion. An insertion of a particular customer i into the work-schedule of a specific vehicle j is feasible only if it does not lead to violation of any service-quality constraints for customer i and for all other customers *already* assigned to vehicle j. The optimization steps deal with minimizing the additional "cost" due to inserting customer i into a vehicle's work-schedule. The cost-function that we use is a weighted sum of disutility to the system's customers (due to excess ride times and to deviations from the most desirable pick-up or delivery times) and of system costs as represented by a function that quantifies the "consumption" of available vehicle resources

Consider now a case in which there are N (advance-request) customer demands for service and m available dial-a-ride vehicles. ADARTW begins by indexing customers in the order of their "earliest pick-up times", EPT_i (i = 1, 2, ... , N), i.e., according to the earliest time at which they are expected to be available for a pick-up. The first customer in the sequence is the one with the smallest EPT_i (Section 3.11 shows how the EPT_i are computed).

The algorithm then processes the first, second, third, etc. customers in the list, one at a time (see also below), and assigns each customer to a vehicle until the last customer is exhausted The processing of customer i goes as follows :

Step 1 : For each vehicle j (j = 1, 2, ... , m), in which a customer is already assigned.

(i) Find all the possible ways in which customer i can be inserted into the work-schedule of j (details in Section 3 11).

(ii) Find the insertion of customer i into the work-schedule of vehicle j that results in minimum additional cost (details in Section 3.12). Call this additional cost, $COST_j$

Step 2 . If it is infeasible to insert i into the work-schedule of any vehicle, then add new vehicle in the system and assign the customer to it; otherwise assign i to a vehicle j^* for which $COST_{j^*} \leq COST_j$ for all j = 1, 2, ... , m.

The above is only the "generic" version of the algorithm. Instead of processing one customer at a time, the user can specify how many (yet unassigned) customers should be considered in Step 1 above. For example, if the number "5" is specified, the top 5 (in terms of their ordering index) unassigned customers will be considered on each occasion as candidates to be assigned next to a vehicle. It should be emphasized that each of the candidates is considered separately in Step 1, so that in Step 2 the one among (the 5, in our example) candidates with the smallest $COST_{j^*}$ will be assigned to vehicle j^*. We can term such a group of candidates as the candidate "pool" from which the next customer for insertion is selected. There exist two possible strategies for bringing unassigned customers into the pool. One strategy would be to bring a new customer into the pool every time a customer in the pool is inserted into some vehicle's work-schedule. In this

way, a pool always maintains the same number of candidate customers in it. A customer will leave the pool either when he is selected for next insertion in Step 2 or when it is infeasible for any vehicle to carry him. Such a strategy is termed *immediate-refill*. An alternative is to let the candidate pool become smaller and smaller as customers in the pool are inserted, one at a time, to vehicle work-schedules. When the pool is finally exhausted, refill it with the next batch of customers in the list. This strategy is termed *periodic-refill*. Kapil [5] shows that the immediate refilling is better than periodic refilling and best pool size is one

This multi-candidate option is provided for the purpose of improving the performance of the algorithm by making it less "myopic", i.e , by giving it an opportunity to select among more than one customer for the next assignment. The penalty, of course, is that as the number of candidate that are considered each time increases the efficiency of the algorithm decreases

3.11 SEARCH FOR FEASIBLE INSERTIONS

We now turn to an outline of the steps taken to identify feasible insertions of customers into vehicle work-schedules. A notion which plays an important role in this respect is that of a "schedule block". This can be illustrated through Figure 3.2, which shows part of the work-schedule of a vehicle. A schedule block is a period of continuous active vehicle time between two successive periods of vehicle slack time. A schedule block always begins with an empty vehicle starting (either from a depot or after an inactive period) on its way to pick up a customer and ends with a period of slack time or at the end of the vehicle's entire work-schedule. In the case of a schedule block, a

vehicle might become empty several times before the schedule block ends. Associated with a schedule block is a "schedule sequence" indicating the sequence of stops in the block and a "time schedule" indicating the time when each stop is scheduled to take place. For example, in Figure 3.2, the schedule sequence associated with the middle schedule block is (+k, +m, -m, +m, -k, -n) while the time schedule is (APT_k, APT_m, ADT_m, APT_n, ADT_k, ADT_n)

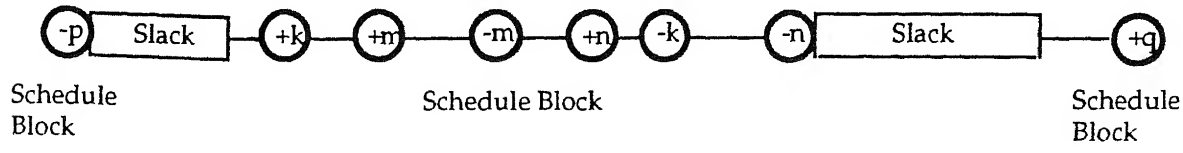


Figure 3.2

Suppose now that we wish to examine whether a yet-unassigned customer i can be inserted into the work-schedule of vehicle j. The objectives of the search for feasible insertions are :

(i) To identify new feasible schedule sequences

(ii) For each of the feasible schedule sequences to obtain a feasible time schedule.

ADARTW accomplishes these objectives by examining systematically and efficiently all possible schedule sequence associated with each and every schedule block on the work-schedule of vehicle j For example, with respect to the middle schedule block of Figure 3.2, the possible schedule sequence involving the insertion of customer i are (+i, -i, +k, +m, -m, +n, -k, -n), (+i, +k, -i, +m, ... , -m), ... , (+k, +m, ... , -n, +i, -i). In all if there are d stops already on a schedule block, there are (d+2)(d+1)/2 possible schedule sequences that involve the insertion of a new customer into that schedule block, while adhering to the constraint that the "+i" stop must precede the stop "-i". In

view of the maximum ride time and maximum time-deviation constraints of our version of DARP (relation (3 1)-(3 3)) only some (and possibly none) of the above possible sequences may be feasible.

More schedule sequences are possible if we consider the insertion of stop "+i" and stop "-i" into different - not necessarily consecutive - schedule blocks. Such insertions will result in merging all the schedule blocks between (and including) the ones where the pick-up and delivery are inserted. ADARTW can consider both types of insertions, but uses different methods (see below) to test the feasibility of schedule sequences formed.

It should be noted that, in addition to inserting customer i into one of the already existing schedule blocks of any vehicle j, ADARTW will, naturally, consider creating an entirely new schedule block for vehicle j, as shown in Figure 3.3, in order to accommodate customer i. For example, the first customer ever assigned to a vehicle will obviously always create a new schedule block. Such new schedule blocks are added to the list of existing schedule blocks to which ADARTW attempts to add new customers through the insertion procedure.

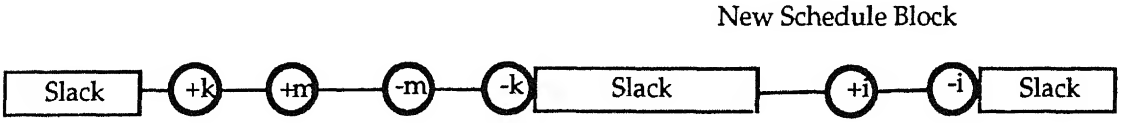


Figure 3.3

A Fast Screening Test For Insertions Within The Same Schedule Block

To facilitate the feasibility search, ADARTW includes a procedure that increases greatly its efficiency and is fundamental to its viability in dealing

with large-scale problems. This procedure involves defining two time windows for each customer as follows :

For DPT-specified customer, we define :

$$EPT_1 = DPT_1 \tag{3.5}$$

$$LPT_1 = EPT_1 + WS_1 \tag{3.6}$$

$$EDT_1 = EPT_1 + DRT_1 \tag{3.7}$$

$$LDT_1 = LPT_1 + MRT_1 \tag{3 8}$$

Note that (3.5) and (3.6) contain the same information as (3.2).

Similarly, for each DDT-specified customer, we define ·

$$LDT_1 = DDT_1 \tag{3 9}$$

$$EDT_1 = DDT_1 - WS_1 \tag{3.10}$$

$$LPT_1 = LDT_i - DRT_1 \tag{3.11}$$

$$EPT_i = EDT_i - MRT_i \tag{3 12}$$

(3.9) and (3.10) contains the same information as (3 3))

For any customer i, whether DPT- or DDT-specified, a set of necessary, but not sufficient conditions for feasibility is then provided by (3.13) and (3.14) :

$$EPT_1 \leq APT_i \leq LPT_i \tag{3.13}$$

$$EDT_i \leq ADT_i \leq LDT_1 \tag{3.14}$$

It should be noted that LPT_i need not be smaller than EDT_1.

The conditions (3.13) and (3 14) are particularly convenient to work with. The quantities EPT_i, LPT_i, EDT_i and LDT_i computed for all customers i = 1, 2, . , N define "fixes" on the time axis within which the customer must be picked up and delivered To understand the usefulness of these fixes let us return to the problem of checking the feasibility of inserting customer i into a particular schedule block "p" of vehicle j.

Let us consider such a schedule block and let us index the successive stops on the schedule sequence with the subscript r = 1, 2, . , d. Note that from (3.13) and (3.14) we have an upper and lower bound for each element in the time schedule associated with our schedule block ((3.13) providing the bounds if the entry is an APT and (3.14) if the entry is an ADT).

For convenience let us now drop the indication of whether a particular stop on the schedule block is a pick-up or a delivery and use ET_r, AT_r and LT_r to denote earliest, actual (scheduled) and latest time, respectively, for stop r. For instance, in Figure 3.2, we would indicate EPT_k, APT_k, LPT_i by ET_1, AT_1 and LT_1, respectively, and EDT_m, ADT_m, LDT_m by ET_3, AT_3 and LT_3, respectively, for the middle schedule block.

In ARDTW, we compute and store four statistics for each stop r (r = 1, ... , d) on each schedule block, defined as follows :

$$BUP_r = \min \left[\min_{1 \le t \le r} (AT_t - ET_t), SLACK_p \right] \tag{3.15}$$

$$BDOWN_r = \min_{1 \le t \le r} (LT_t - AT_t) \tag{3.16}$$

$$AUP_r = \min_{r \le t \le d} (AT_t - ET_t) \tag{3.17}$$

$$ADOWN_r = \min \left[\min_{r \le t \le d} (LT_t - AT_t), SLACK_{p+1} \right] \tag{3.18}$$

where $SLACK_p$ and $SLACK_{p+1}$ denote, respectively, the duration of the slack period immediately preceding and immediately following the schedule block p in question

There is a very real intuitive meaning associated with the four quantities defined by (3.15)-(3.18). Specifically, BUP_r ($BDOWN_r$) represents the maximum amount of time by which every stop preceding but not including stop r+1 can be advanced (delayed) without violating the time-window constraints. Similarly AUP_r ($ADOWN_r$) represents the maximum amount of time by which every stop following but not including stop r - 1 can be advanced (delayed). Essentially, the quantities BUP, BDOWN, AUP and ADOWN indicate by how much, at most, each segment of a schedule block (e.g., the segment that precedes the pick-up of the inserted customer i) can be displaced in order to accommodate an additional customer i.

The uses of the four statistics defined at each stop for checking the feasibility of an insertion differ depending on where in the schedule block the pick-up and delivery of the new customer are inserted. ADARTW divides all insertions into four basic cases , which account for the total of (d+2)(d+1)/2 combinations mentioned earlier :

(i) Both the pick-up and delivery of customer i are inserted at the end of the *last* schedule block, i.e., they become the last two stops in the vehicle's work-schedule. We note that this is the only case where a new schedule block can be created.

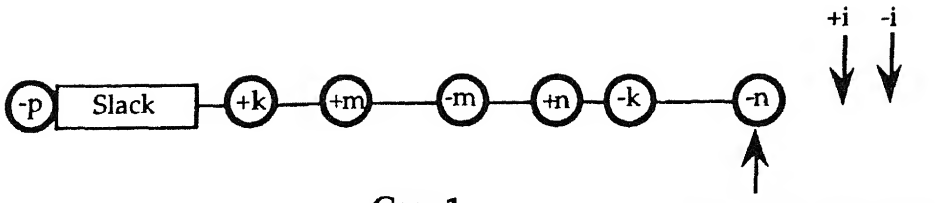


Figure 3.4

(ii) Both the pick-up and delivery of customer i are inserted between two consecutive stops on the schedule block. This includes the insertions of pick-up and delivery immediately before or after a slack period

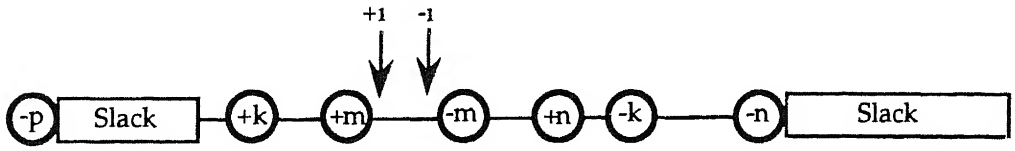


Figure 3.5

(iii) The pick-up of customer i takes place somewhere within the *last* schedule block while his delivery is inserted at the *end* of the vehicle's work schedule.

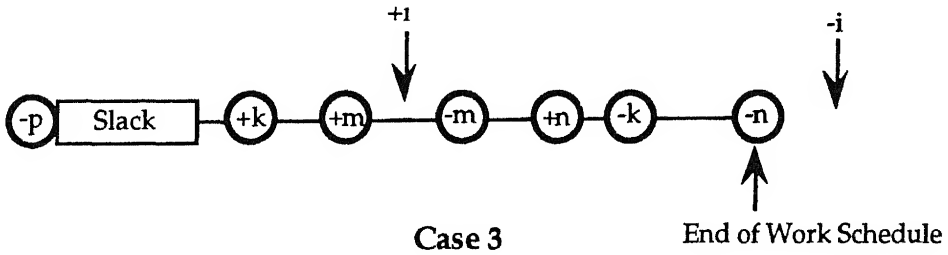
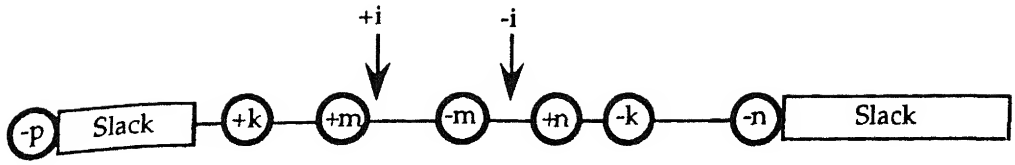


Figure 3.6

(iv) The pick-up and delivery of customer i are separated by at least one other stop and the delivery of i is not the last stop in the vehicle's work-schedule.



Case 4

Figure 3.7

For The details of the algorithm's logic see J J. Jaw[4]. Besides checking for violations of the time-window constraints, we have to check that no maximum-ride-time constraints are violated for the newly inserted customer and for the customers already in the schedule block. This can be done very easily by scanning through the list of these customers and comparing the respective actual ride-time and maximum available rime-time. Finally, vehicle loads at each stop between the inserted pick-up and delivery of customer i are checked so that vehicle capacity is not exceeded.

Feasibility Test for Insertions into Different Schedule Blocks

The insertion of the pick-up and delivery of customer i into different schedule blocks requires the estimation of all the slack time contained between these schedule blocks since, otherwise, the vehicle would wait idly with customer i on board. Let us consider the case of inserting the pick-up of customer i somewhere in schedule block k and the delivery of customer i into schedule block k+n ($n \geq 1$). We limit our search for feasible insertions to those that will not affect other customers on schedule blocks q, $q < k$ or $q > k+n$. Consequently, as a priori condition for an insertion to be considered in our search is :

$$\Delta P_i + \Delta D_i \leq \sum_{j=k}^{k+n+1} SLACK_j \qquad (3.19)$$

where ΔP_i (ΔD_i) is the extra vehicle time required to serve the pick-up (delivery) point of customer i

The algorithmic approach works as follows :

Form a new schedule sequence by merging schedule blocks k to k+n and with customer i inserted. Let d denote the number of stops in the new schedule sequence Construct an earliest time schedule (without considering the time constraints at each stop) for the new sequence with all the slack time eliminated, i.e., by taking $SLACK_k = 0, SLACK_{k+1} = 0, \ldots, SLACK_{k+n} = 0$. Let us denote this tentative time schedule by "T." For every stop r (r = 1, 2, ..., d) in the new schedule sequence, compute two statistics, R_r and A_r :

$$R_r = |Min\ (T_r - ET_r, 0)| \tag{3.20}$$

$$A_r = LT_r - T_r \tag{3.21}$$

where T_r is the visiting time of stop r in time schedule T.

R_r denotes the minimum time by which the visit of stoop r, T_r, should be delayed for it to fall within the time window of stop r. A_r denotes the maximum time by which T_r can be delayed for it not to violate the time window constraint of stop r. After R_r and A_r are computed for all r in the new schedule sequence, we can define R_{min} and A_{max} for time schedule T as :

$$R_{min} = \underset{\text{all r}}{\text{Max}}\ (R_r) \tag{3.22}$$

$$A_{max} = \underset{\text{all r}}{\text{Min}}\ (A_r) \tag{3.23}$$

R_{min} represents the minimum amount of time by which the time schedule T should be delayed so that every stop in T is visited at or after its earliest feasible time. A_{max} represents the maximum amount of time by which the

time schedule T can be delayed without a stoop being visited later than its latest feasible time. For a schedule sequence to be feasible we must have :

$$R_{min} \leq A_{max} \tag{3.24}$$

(3.24) indicates the fact that the amount of time by which T has to be delayed must be less than or equal to the maximum amount of time that T can be delayed (3.24) is the necessary and sufficient condition for a new sequence to be feasible with respect to satisfying the time window constraints To check for violations of vehicle capacity and customer's maximum-ride-time constraints, a screening test through the new time schedule would suffice. (Such screening can be performed at the same time when R_r and A_r computed.)

We now proceed to discuss the optimization procedure used to choose the most desirable insertion.

3.12 THE OPTIMIZATION PROCEDURE

In order to select all feasible insertions of customer i into the work-schedules of the available vehicles, we use an objective function designed to evaluate the incremental "cost" of each insertion. This cost is taken to be a weighted sum of disutility of system's customers and of operator costs - the latter measured in terms of "consumption" of available vehicle resources.

Assume that it is feasible to insert customer i into the work-schedule of vehicle j. Then the incremental disutility of that insertion to the system's customers consists of the sum of two parts: the disutility of customer i, i.e., the customer being assigned to a vehicle; and the additional disutility

suffered by all *other* customers *already assigned to that vehicle* because of the insertion of customer i The first part (disutility of customer i) is given by

$$DU_i = DU_i^d + DU_i^T \tag{3.27}$$

where DU_i^d = disutility due to deviation from most desired time

$$= C_1 x_i + C_1 x_i^2 \qquad 0 \le x_i \le WS_i \tag{3.28}$$

where WS_i is defined in section 3.9.

and,

DU_i^T = disutility due to excess ride time

$$= C_3 y_i + C_4 y_i^2 \qquad y_i \ge 0 \tag{3.29}$$

In (3.28) and (3.29), C_1, C_2, C_3 and C_4 are user-specified constants and

$$x_i = \begin{cases} APT_i - DPT_i & \text{for DPT-specified customers} \\ DDT_i - ADT_i & \text{for DDT-specified customers} \end{cases}$$

(3.30)

and $\quad y_i = ART_i - DRT_i$ (3.31)

The quadratic terms allow the modeling of situations in which disutilities are believed to increase non linearly with x_i and/or y_i. Clearly, by varying C_1, C_2, C_3 and C_4 (including assigning the value of zero to some of them) many different types of behavior can be represented.

The second part (additional disutility to other customers) is given by :

$$DU_i^0 = \sum_{k \text{ on vehicle } j} (DU_k^{new} - DU_k^{old})$$

(3.32)

where DU_k^{new} and DU_k^{old} are, respectively, the disutilities to customer k after and before the insertion of custo

summation is over all customer k who were already assigned to vehicle j prior to the assignment of customer i.

The incremental cost, VC_i, to the system's operator due to inserting customer i into the work-schedule of some vehicle is quantified in somewhat unusual terms by our objective function. We have,

$$VC_i = C_5 z_i + C_6 w_i + U_i(C_7 x_i + C_8 w_i) \tag{3.33}$$

where C_5, C_6, C_7 and C_8 are user defined constants

z_i is the additional active vehicle time required to serve the customer i

w_i is the change in vehicle slack time due to the insertion

U_i is an indicator of system workload defined as :

$$U_i = \frac{\text{(No. of system customers in } [EPT_i - W_1, EPT_i + W_2])}{\text{(Effective no. of vehicles available in } [EPT_i - W_1, EPT_i + W_2])} \tag{3.34}$$

In (3.34), W_1 and W_2 are user specified constants. For example, if $W_1 = 0, W_2 = 60$ minutes, U_i is equal to the ratio of the number of customers demanding service to the available number of vehicles during the one-hour time period that has the earliest pick-up time of customer i as its mid-point. The word "effective" is used in the denominator of (3.34) to account for the fact that some vehicles may be available for service only for a fraction of the time interval (e.g., two hours) in question - usually because of driver constraints, union rules, etc.

The following should be noted with respect to (3.33) :

(i) The change in vehicle slack time, w_i, can be positive or negative. It will be negative if the insertion means that slack time in the original schedule will now be utilized to serve customer i; it will be positive if,

in order to serve customer i, additional vehicle slack time must be created (this will happen if a new schedule block is created to serve customer i)

(ii) In practice, the cost per unit of time of a vehicle in the "active" state is greater than in the "slack" state (e.g., no fuel consumption in slack state). Therefore, we must have $C_5 \geq C_6$ and $C_7 \geq C_8$.

(iii) Obviously U_1 will be larger during periods of heavy demand Since the total "cost" of an insertion is given by $DU_1 + VC_1$ - i.e. by the sum of (3.27), (3.32) and (3.33) - it is clear that during heavy demand periods, the objective function places more emphasis on the system operator's costs (relative to customers' disutility) than during low demand periods. This is as it should be, since during periods of high utilization vehicle resources are scarcer and it is thus important to "conserve" these resources as much as possible Kapil [5] did the sensitivity analysis of U.

Given this background, the optimization steps of ADARTW can be summarized as follows: Consider customer i who is to be inserted into the work-schedule of one of m available vehicles. Assume that for each vehicle j (j = 1, 2, ... , m) all feasible sequences for inserting customer i into the work-schedule of j have been identified in the manner outlined in Section 3.11. Then, to select the optimum insertion, we use a sequence of three step :

(a) For each and every feasible sequence pertaining to vehicle j, find a time schedule (including the pick-up and delivery times APT_1 and ADT_i for customer i) that minimizes the incremental cost $DU_i + VC_i$. (Note that this requires shifting the feasible time schedule associated with each feasible sequence (obtained in Section 3.11) within its bounds of displacement.)

that this requires shifting the feasible time schedule associated with each feasible sequence (obtained in Section 3.11) within its bounds of displacement.)

(b) Choose the sequence among those examined above which results in the smallest incremental cost. This smallest incremental cost is $COST_{j'}$ the additional cost associated with inserting customer i into the work-schedule of vehicle j as defined in Section 3.10 above.

(c) Finally, assign customer i to vehicle j such that $COST \leq COST_j$ for all j = 1, 2, ... , m

3.13 SIMULATED DATA

In order to gain insights into the performance of different algorithms, we generated a set of customers subscription lists. These lists were based upon combinations of 6 different demand scenarios, and two different sizes of time windows. The set of time windows (20 and 30 minutes) represent a possible range of service quality levels that could be guaranteed by the system operations. We have maintained two types of service quality levels, which will be differentiated according to the time window and ride time constraints.

While generating the data, we have considered the following scenario :

Service Area : 6 x 6 square miles

Distance Metric : Rectilinear

Customer origin/Destination : Uniformly and Independently distributed in the area.

Vehicle origin : Center of the service area

Vehicle destination : Centre of the service area

Vehicle speed · 15 miles/hr

Start time of service : 9 AM

End time of service : 5 PM

Percent of DPT-specified customer . 50%

Time window size : 20 minutes for express service

30 minutes for normal service

Maximum Ride Time : 10 + 1.7 DRT for express service

10 + 1.5 DRT for normal service.

Parameters W_1 and W_2 used in calculation of system operating cost : W_1 = 0, W_2 = 60 minutes.

Number of customers per request is 1, 2 or 3 with the probability 0.7, 0.2, 0.1

We have divided the service duration (from 9 AM to 5 PM) into 4 equal parts of two hour duration, i.e., [9 AM, 11 AM), [11 AM, 1 PM], [1 PM, 3 PM] and [3 PM, 5 PM], regarding the generation of Desired pick up and delivery time. For each instance 75% of the Desired pickup or delivery time of the customer will fall into one of time slots. In this way the service timings requested are spread evenly through the duration in which service is provided.

Each vehicle may have different capacities, but for our experiment purpose we have assumed capacity equal to 8 for all the vehicles. We have also assumed that vehicle is available for all the 24 hours. The parameters value we have taken to get results are as follows:

customer pool size = 1 (immediate refilling)

C1 = 3, C2 = 0, C3 = 0.1, C4 = 0, C5 = 0, C6 = 0, C7 = 1.2,

C8 = 0.7

In immediate request version, the program runs after every 15 minutes. The U value for the new trip request is 0.1.

With the above assumptions, we have prepared the subscription list of different sizes. These are 100, 200, 300, 400, 500 and 1000 customers in each subscription list. In turn for each size we have generated 5 instances of each sizes. All these instances are having only many-to-many type of customer requests. We also generated instances of 1000 customers having 25%, 50%, 75%, and 100% many-to-few type of customer requests.

CHAPTER 4: DIFFERENT APPROACHES TO SOLVE ADART PROBLEM

4.1 INTRODUCTION

A great amount of effort have been put in by various researchers to handle advance trip requests optimally. J. J. Jaw [4] worked on different strategies to handle advance requests We used his algorithms as a base of our work. This chapter describes about the different approaches we used to solve ADART problem We take results of different approaches on simulated data The run time shown in Figures for approaches designed by us are the run time of ADARTW algorithm plus the run time required to run these approaches The discussion consists of five parts. In Section 4.2 the different improvement strategies for advance requests are discussed. Section 4.3 discusses about the strategies to handle immediate requests. Section 4.4 discusses the improvement strategies for immediate request. Section 4.5 discuses about cancellation of customer requests, and Section 4.6 discusses about private taxi simulation, and comparison of its performance with ADART taxi service. Section 4 7 describes the avenues for future work

4.2 IMPROVEMENT STRATEGIES FOR ADVANCE REQUEST VERSION

We used the algorithms designed by J. J. Jaw [4] and Kapil [5] for finding the initial feasible solution for advance trip requests. These algorithms gives a feasible solution, but not the best solution, because these algorithms are heuristic algorithms not exact algorithms. There is a lot of scope of

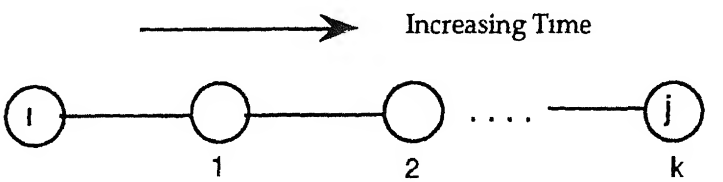
improvement in the initial solution. So we designed different algorithms to improve the initial solution. Improvements can be local or global. In local improvement, we try to improve the route of a vehicle by changing the order of customers' pick-ups and deliveries within a route In global improvement, the exchange of customers among different routes take place. The different local and global strategies designed by us are as follows :

(i) Improvement by Swapping Nodes

This is a local improvement method. In this we try to improve the route of a vehicle by swapping the nodes of a route We restricted ourselves only to the feasible swaps The feasible swaps are those swaps which do not create infeasibility in the route.

For swapping we can choose two adjacent nodes and also the nodes those are far apart. There may exist more than one node between the nodes for which we are doing swapping. If the nodes chosen for swapping are i and j then they must satisfy the following two conditions :

- if j is a delivery node then its pickup node should be after j
- if i is a pickup node then its delivery node should be after j.



The parameter **k** is called the swapping range. If the nodes satisfy the conditions above then we swap and if we do not get a feasible route after swapping then we can try to make it feasible as shown in Figure A.1. If the route can be made feasible then the swap is permissible otherwise not. We used two strategies for improvement with the help of swapping.

(a) First Swap

The Figure A 2 shows an overall view of this strategy. In this, we first try to find feasible swap (*first swap*) in the original route, that can decrease the objective function value of the route. After finding such swap, we perform this swap on the original route to get an improved route. Then we try to find feasible improving swap in the improved route. We keep on doing this as long as we keep getting feasible improving swap in the route

(b) Best Swap

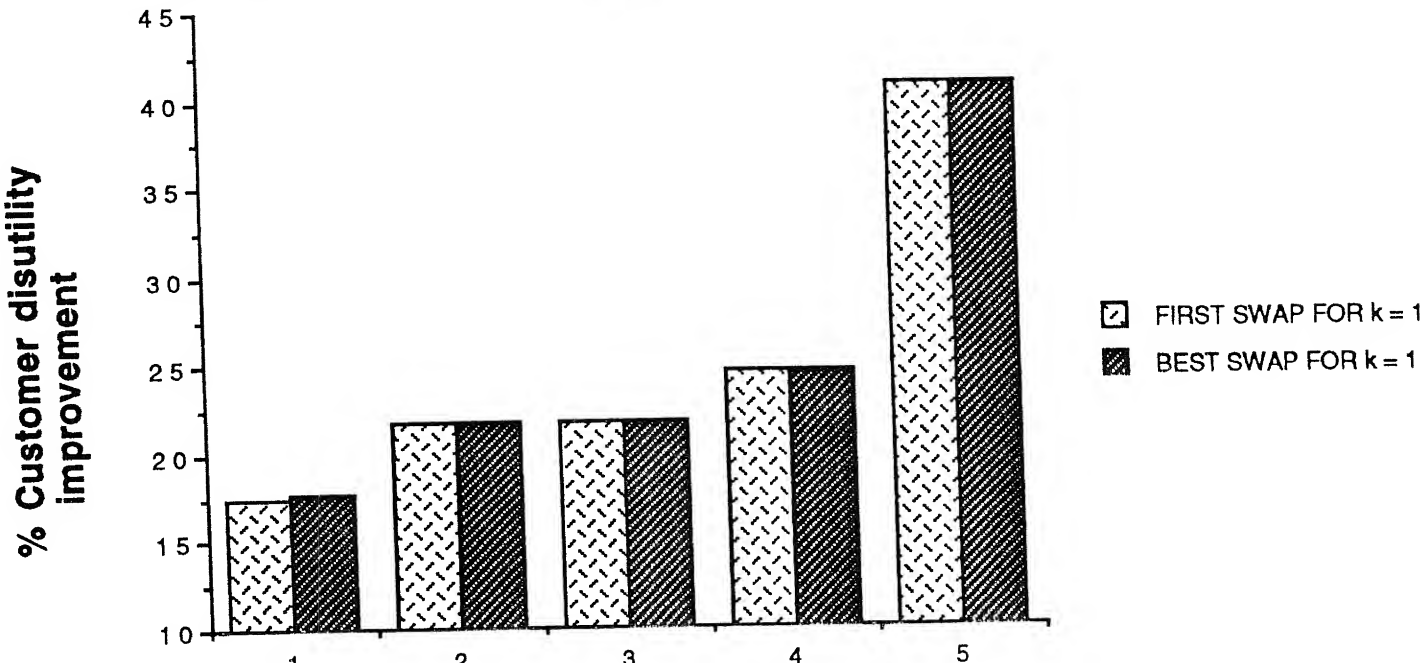
The Figure A 3 shows an overall view of this strategy In this, we first try to find all feasible swaps in the route, those can decrease the objective function value of the route. We choose the swap (*best swap*) which can decrease the objective function value by the maximum amount. We perform such a swap on the original route to get an improved route. Then we try to find best possible swap in the improved route. We keep on doing this until we keep getting such swaps in the route

Computational results

We observe that the best swap strategy is better than the first swap strategy (as shown in Figure 4 1). Our explanation for such behavior is that the best swap strategy uses more greedy approach than the first swap strategy, and that is the reason why best swap strategy improves more. (Sometimes due to the tight window and ride time constraints, the *first swap* eliminates the possibility of swapping nodes which can improve the objective function by a larger amount, which is not the case with *best swap*.)

When we swap the nodes, which are not necessary adjacent (i.e., $k > 1$), we find that the result improves. For $k = 2$, the results are best as shown in Figures 4.2 and 4.3. The reason for such behavior is that when we increase

Comparison between the % customer disutility improvement done by first swap and best swap strategy for k = 1



Best instances of different customers size
(1 = 400, 2 = 100, 3 = 100, 4 = 100, 5 = 300)
Figure 4.1

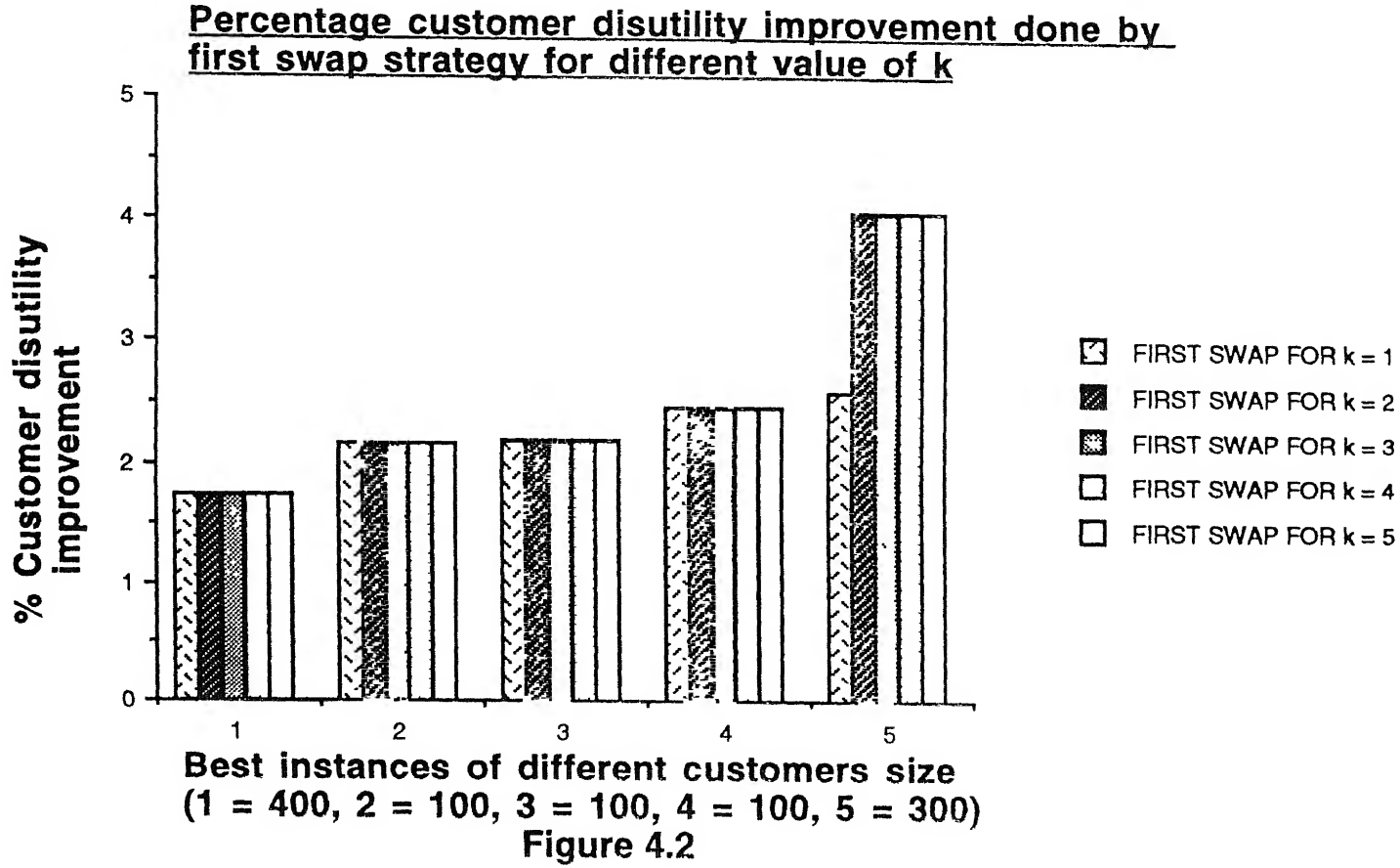

Percentage customer disutility improvement done by first swap strategy for different value of k
% Customer disutility improvement
0
1
2
3
4
5
1
2
3
4
5
Best instances of different customers size
(1 = 400, 2 = 100, 3 = 100, 4 = 100, 5 = 300)
FIRST SWAP FOR k = 1
FIRST SWAP FOR k = 2
FIRST SWAP FOR k = 3
FIRST SWAP FOR k = 4
FIRST SWAP FOR k = 5

Figure 4.2

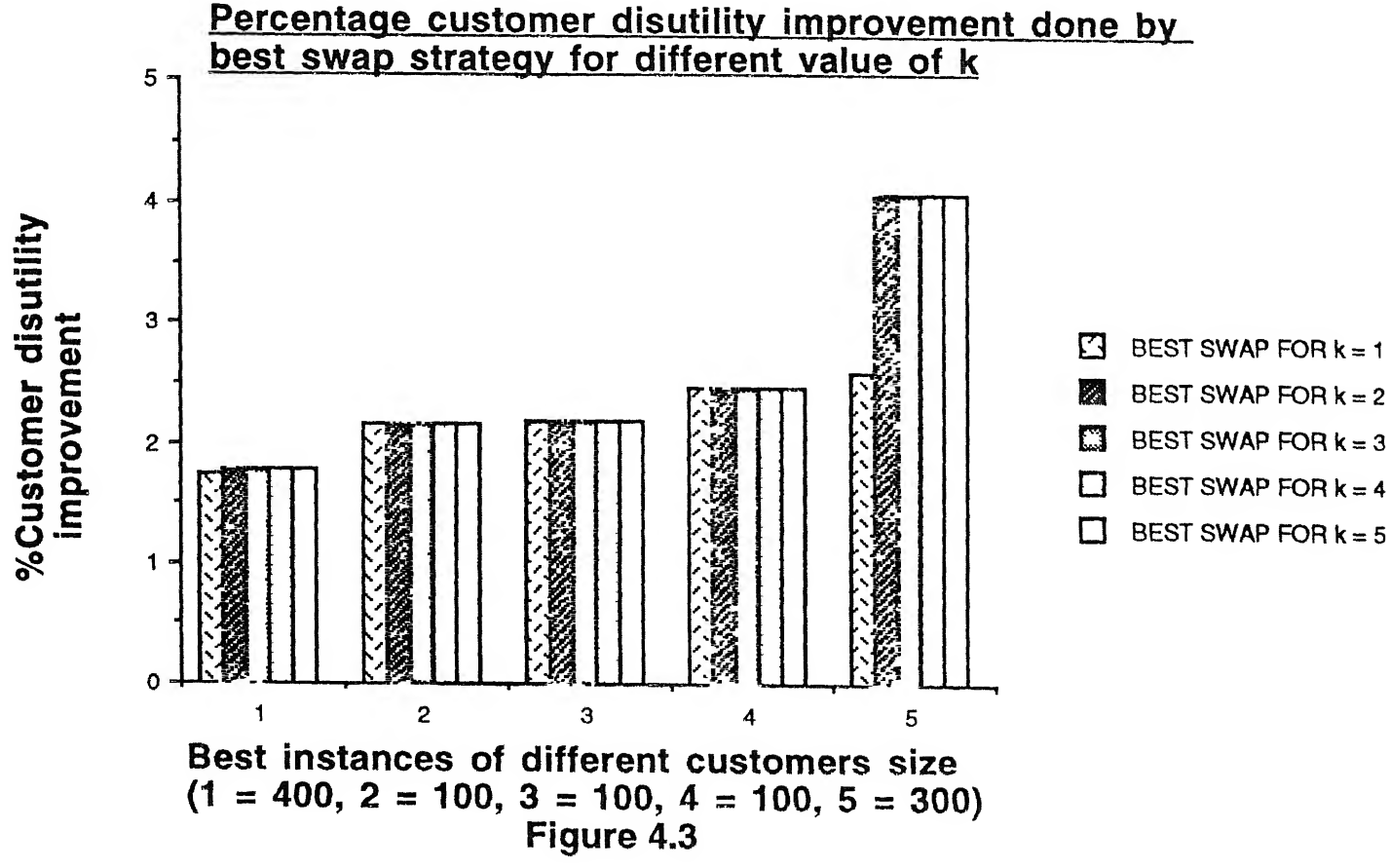

Percentage customer disutility improvement done by best swap strategy for different value of k
%Customer disutility improvement
0
1
2
3
4
5
1
2
3
4
5
Best instances of different customers size
(1 = 400, 2 = 100, 3 = 100, 4 = 100, 5 = 300)
BEST SWAP FOR k = 1
BEST SWAP FOR k = 2
BEST SWAP FOR k = 3
BEST SWAP FOR k = 4
BEST SWAP FOR k = 5

Figure 4.3

the value of k from 1 to 2, the number of feasible swaps increase, which in turn given us better results. But due to tight time window constraints the further increase in the value of k does not increase the feasible swaps, so increasing the value of k does not improve solution further.

We take results to see the effect of problem size on strategy. We take five problems of 100, 200, 300, 400, 500, and 1000 customers each. We observe that the percentage improvement in customer disutility are approximately the same in all instances as shown in Figure 4.4.

The time required to run this algorithm is far less than the run time of ADARTW algorithm, as shown in Figure 4 5 The run time taken by the first swap and best swap strategies are approximately the same. Our explanation for this is that most of the time the feasible swaps, which can improve the objective function are too few, and in most of the cases, such type of swaps found is only one

The CPU time required to run improvement algorithm for different values of k takes approximately same time as shown in Figures 4.6 and 4.7. This implies that even the feasible cases of swapping increase, but most of the cases do not satisfy the preliminary constraints of swapping. So the number of feasible cases do not grow much with the increase in the value of k.

The effect of problem size on run time are shown in Figure 4.8. This Figure shows that the increase in the run time is proportional to the increase in the problem size.

This algorithm does not improve the system operating cost because swaps do not significantly change the active time and wait time of the vehicle.

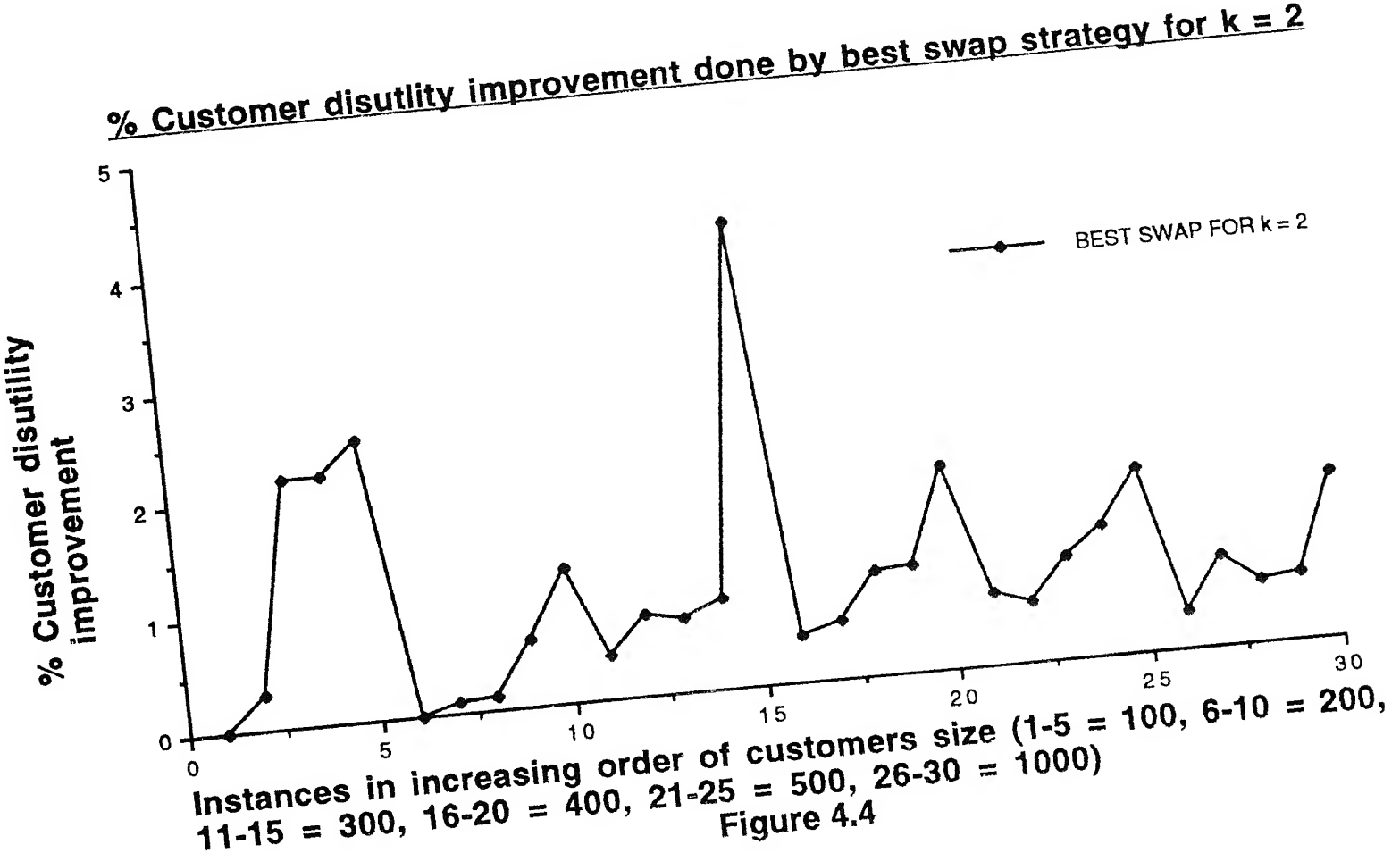

% Customer disutlity improvement done by best swap strategy for k = 2
% Customer disutility improvement
BEST SWAP FOR k = 2
0
1
2
3
4
5
0
5
10
15
20
25
30
Instances in increasing order of customers size (1-5 = 100, 6-10 = 200, 11-15 = 300, 16-20 = 400, 21-25 = 500, 26-30 = 1000)

Figure 4.4

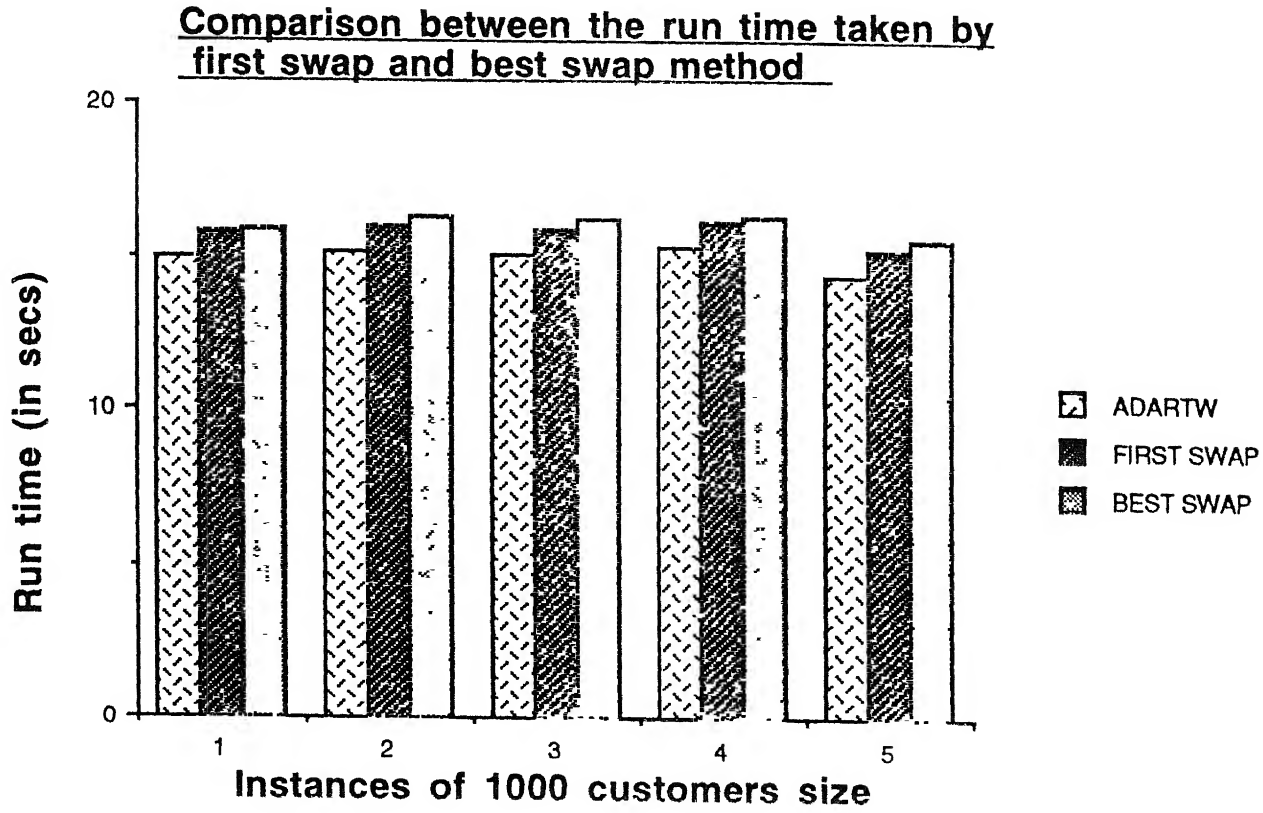

Comparison between the run time taken by first swap and best swap method
20
10
0
Run time (in secs)
1
2
3
4
5
Instances of 1000 customers size
ADARTW
FIRST SWAP
BEST SWAP

Figure 4.5

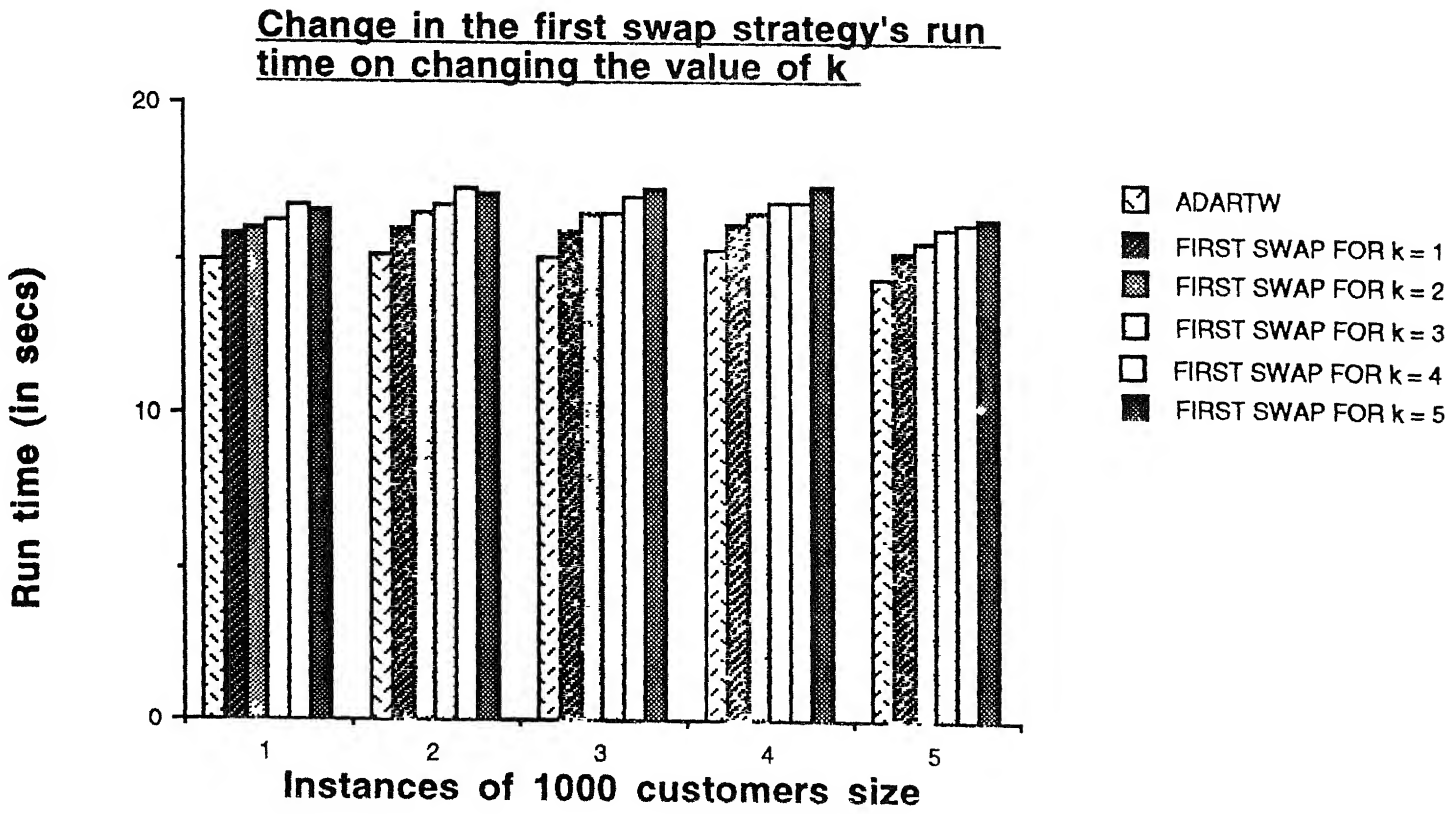

Change in the first swap strategy's run time on changing the value of k
Run time (in secs)
20
10
0
1
2
3
4
5
Instances of 1000 customers size
ADARTW
FIRST SWAP FOR k = 1
FIRST SWAP FOR k = 2
FIRST SWAP FOR k = 3
FIRST SWAP FOR k = 4
FIRST SWAP FOR k = 5

Figure 4.6

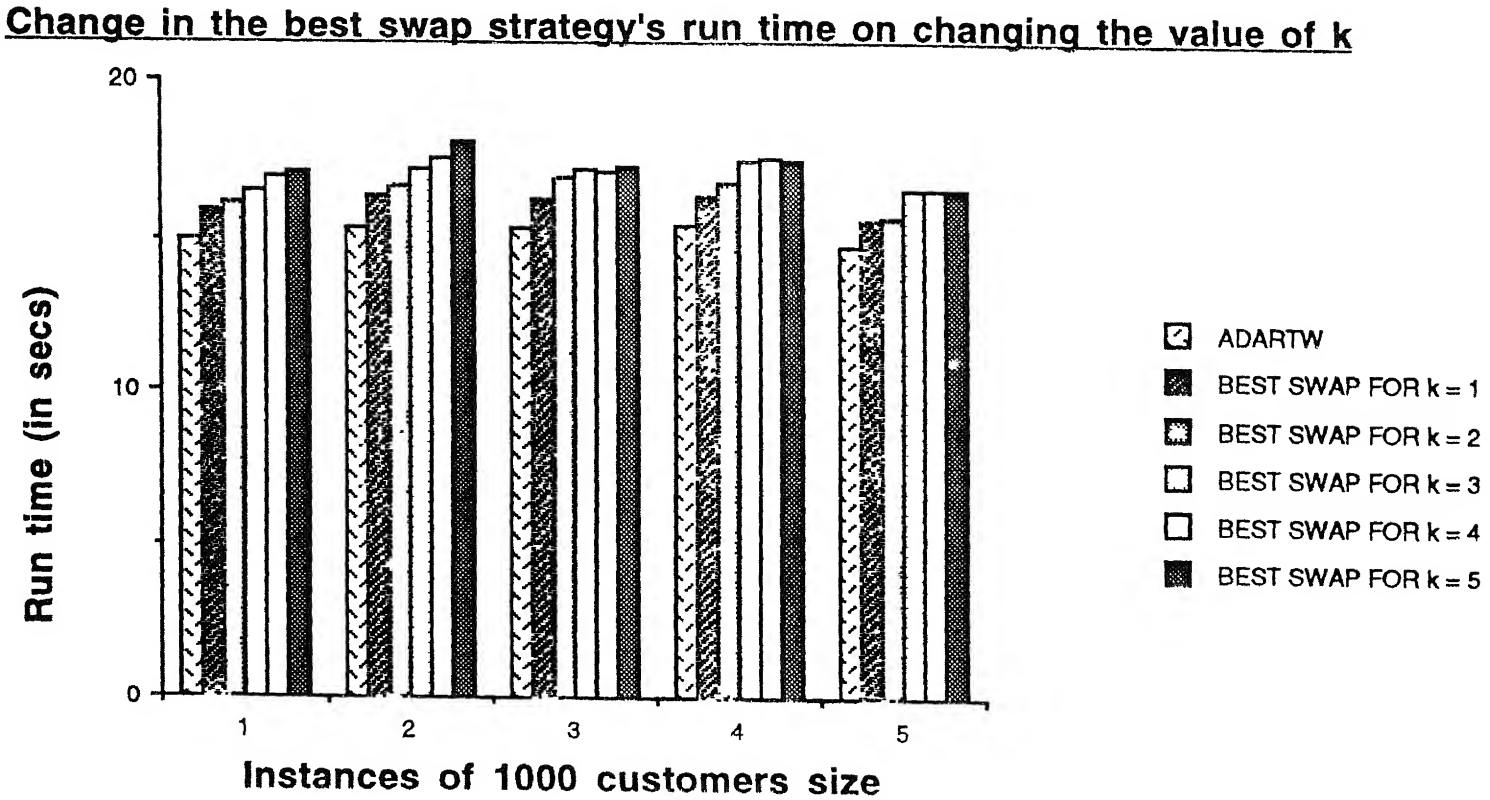

Change in the best swap strategy's run time on changing the value of k
20
10
0
Run time (in secs)
1
2
3
4
5
Instances of 1000 customers size
ADARTW
BEST SWAP FOR k = 1
BEST SWAP FOR k = 2
BEST SWAP FOR k = 3
BEST SWAP FOR k = 4
BEST SWAP FOR k = 5

Figure 4.7

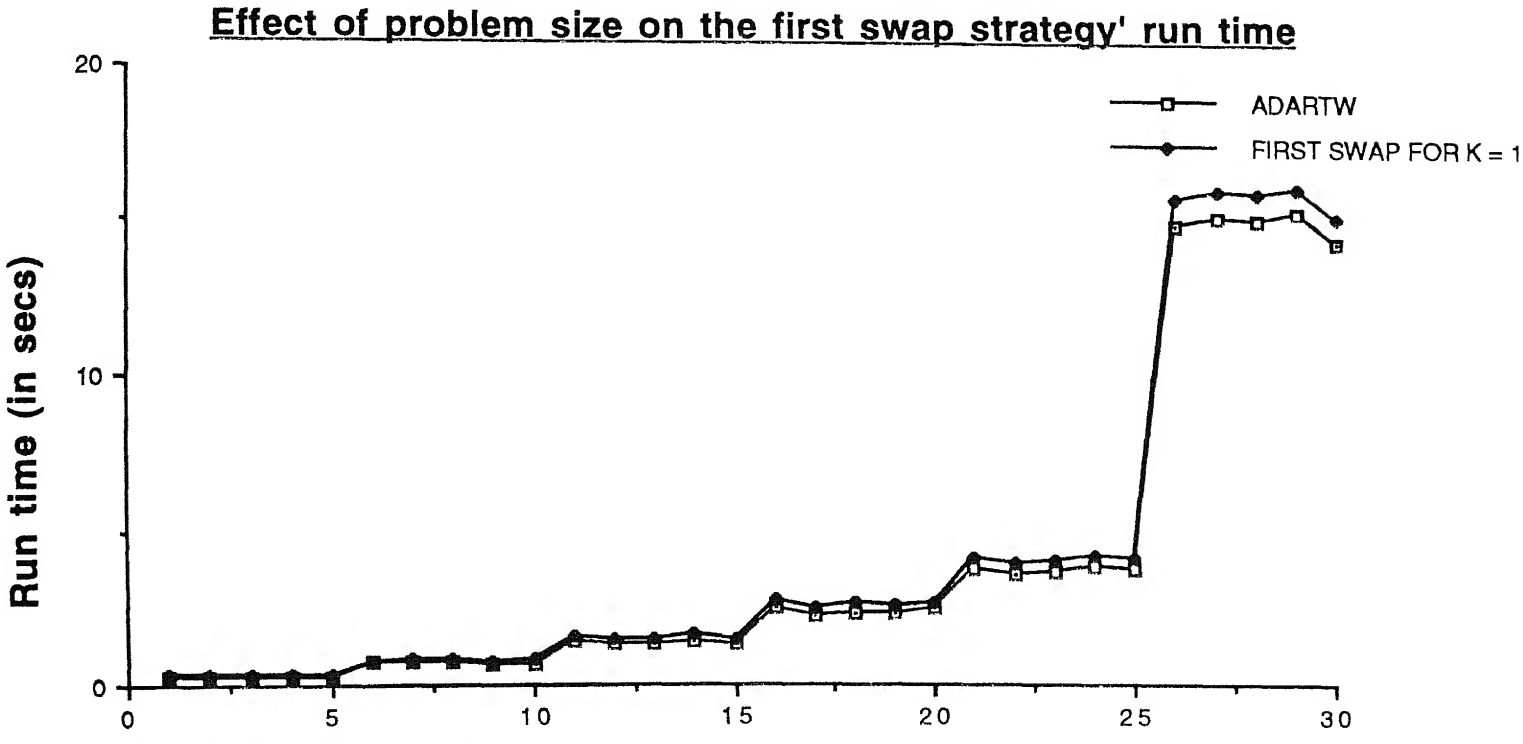

Effect of problem size on the first swap strategy' run time
ADARTW
FIRST SWAP FOR K = 1
Run time (in secs)
20
10
0
0
5
10
15
20
25
30
Instances in increasing order of customers size (1-5 = 100, 6-10 = 200, 11-15 = 300,16-20 = 400, 20-25 = 500, 26-30 = 1000)

Figure 4.8

(ii) Improvement by Enumeration

This is also a local improvement method. The Figures A.4 and A.5 shows the working of this strategy. The initial solution gives us the sets of customers served by different vehicles. After partitioning customers among vehicles, our aim is to find a good feasible route for each set of customers. In this approach, we first enumerate feasible routes for a set of customers, and then select the best enumerated route as the improved route.

If we enumerate more paths, we will get more improvement, but the enumerating routes takes more time Sometimes the time is more crucial, and sometimes improvements are more crucial So in our algorithm we used a parameter x to control the total enumerated routes. If x is very high, then enumeration is called complete enumeration, because for high values of x the algorithm enumerates all feasible routes. When x is small, then enumeration is called biased enumeration because the algorithm does not try all feasible ways of inserting customer to enumerate all routes. In biased enumeration, we try only x best way of inserting a customer to enumerate different routes.

Computational results

For complete enumeration, we find that as the number of customers per vehicle increase, the total enumerated routes per vehicle increase exponentially as shown in Figures 4.9 and 4.10. Our explanation for this is that as the number of customers per vehicle increase, the total feasible ways to insert a customer increase, so the total enumerated routes increase.

As the problem size increase, the run time taken for complete enumeration increase is shown in Figure 4.11.

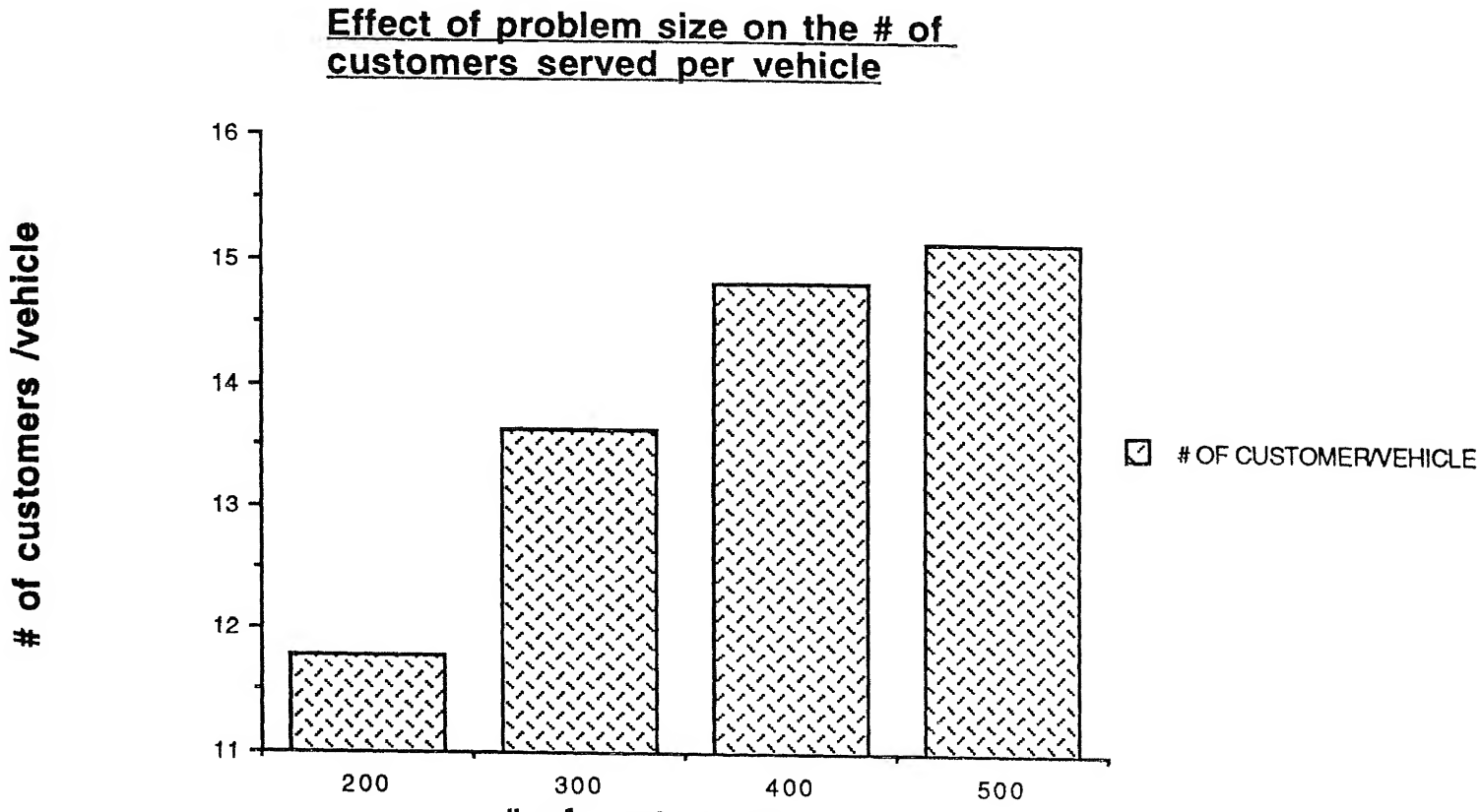

Effect of problem size on the # of customers served per vehicle
# of customers /vehicle
16
15
14
13
12
11
200
300
400
500
# of customers
OF CUSTOMER/VEHICLE

Figure 4.9

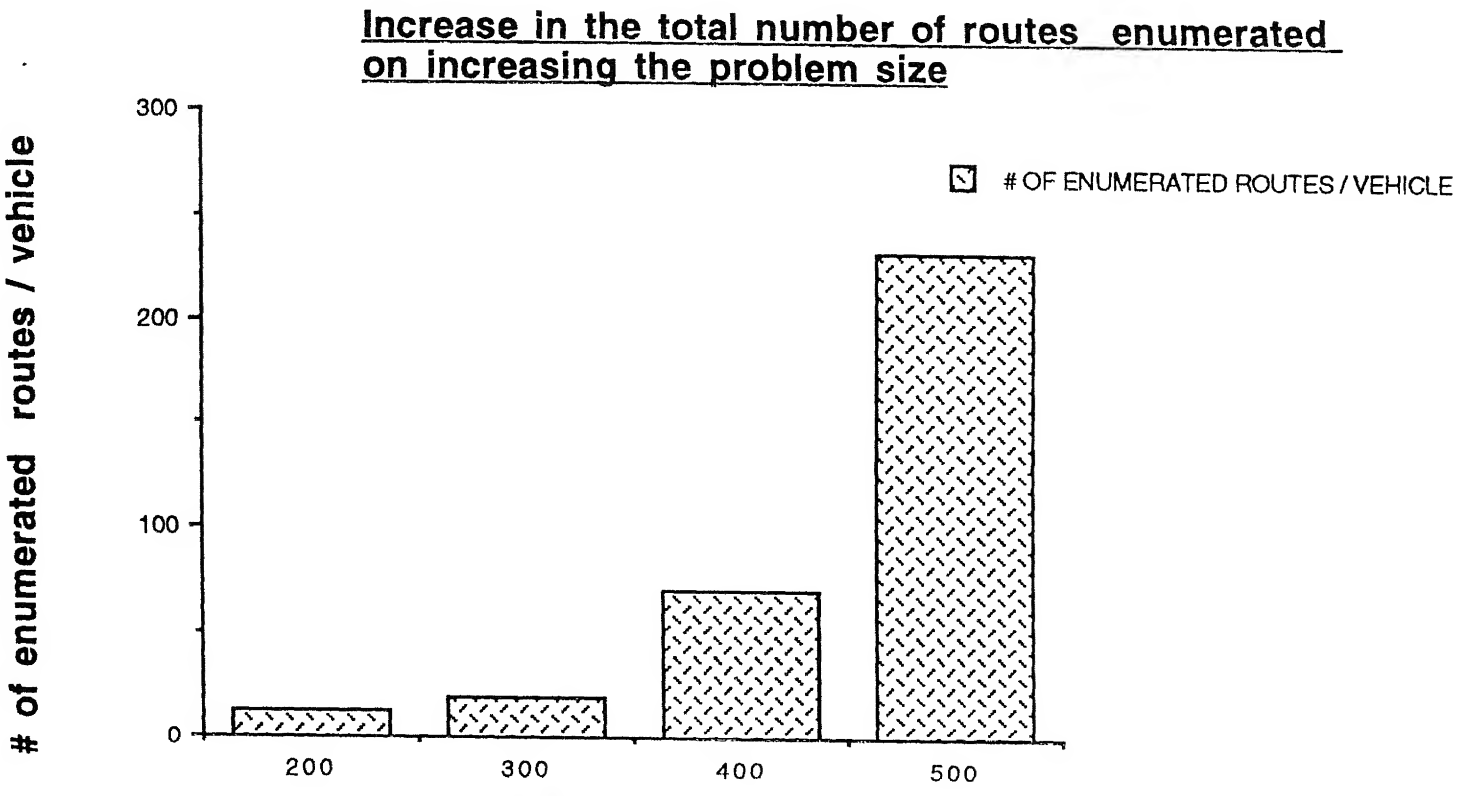

Increase in the total number of routes enumerated on increasing the problem size
OF ENUMERATED ROUTES / VEHICLE
of enumerated routes / vehicle
300
200
100
0
200
300
400
500
# of customers

Figure 4.10

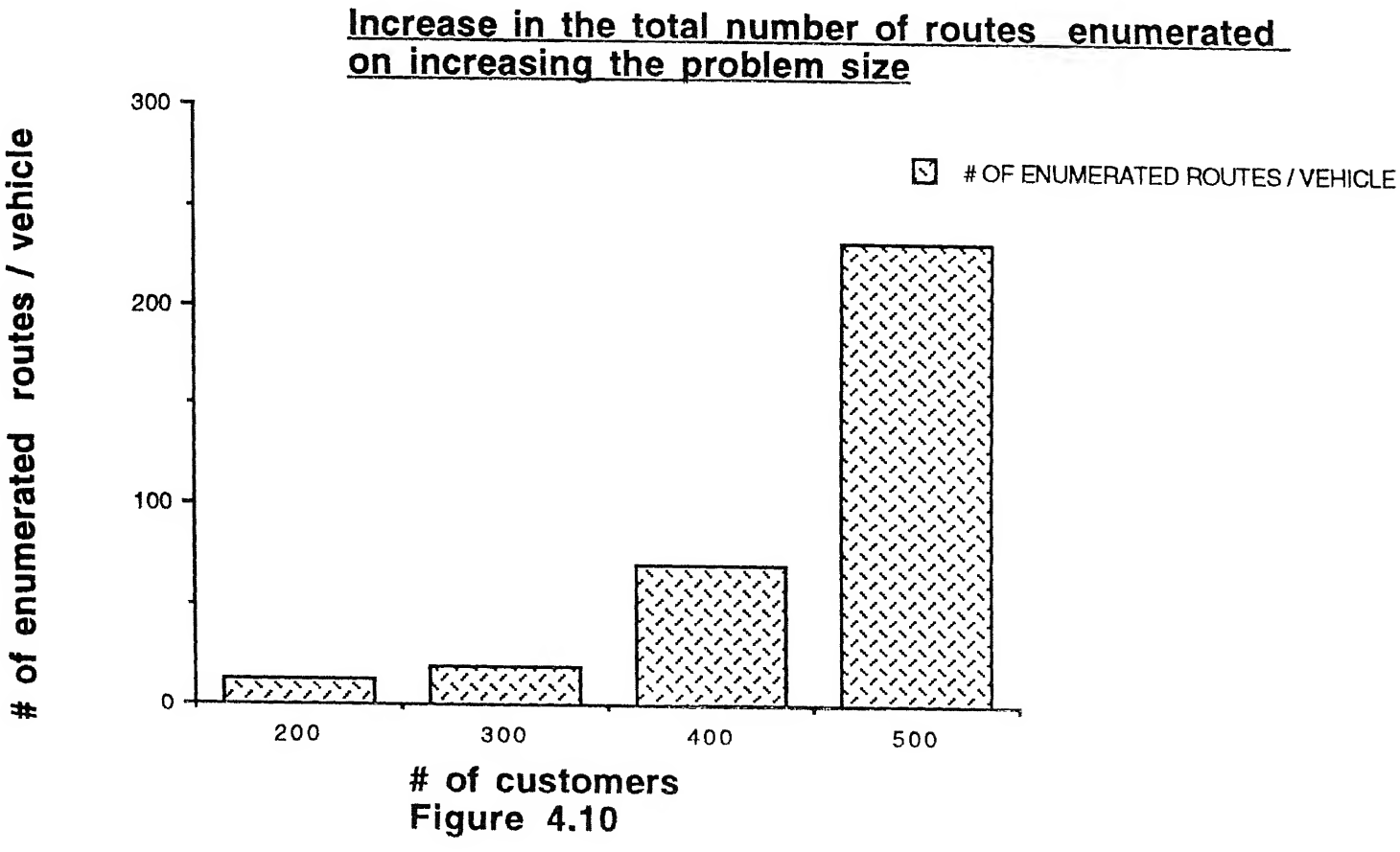

Increase in the total number of routes enumerated on increasing the problem size
OF ENUMERATED ROUTES / VEHICLE
of enumerated routes / vehicle
300
200
100
0
200
300
400
500
# of customers

Figure 4.10

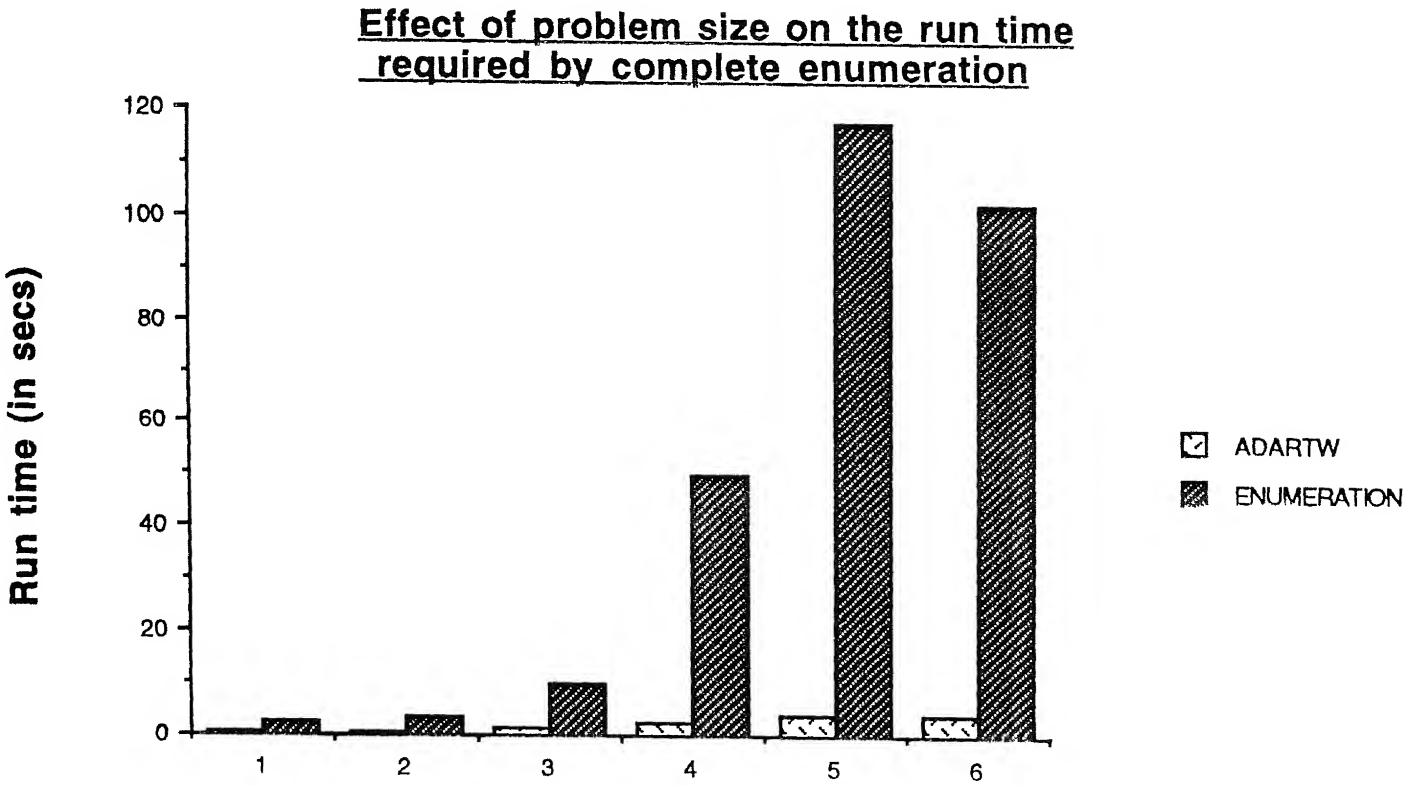

Effect of problem size on the run time required by complete enumeration
Run time (in secs)
120
100
80
60
40
20
0
1
2
3
4
5
6
Instances of different customers size (1 = 100, 2 = 200, 3 = 300, 4 = 400, 5 = 500, 6 = 500)
ADARTW
ENUMERATION

Figure 4.11

Comparison between the total objective function value obtained by biased enumeration and best swap method

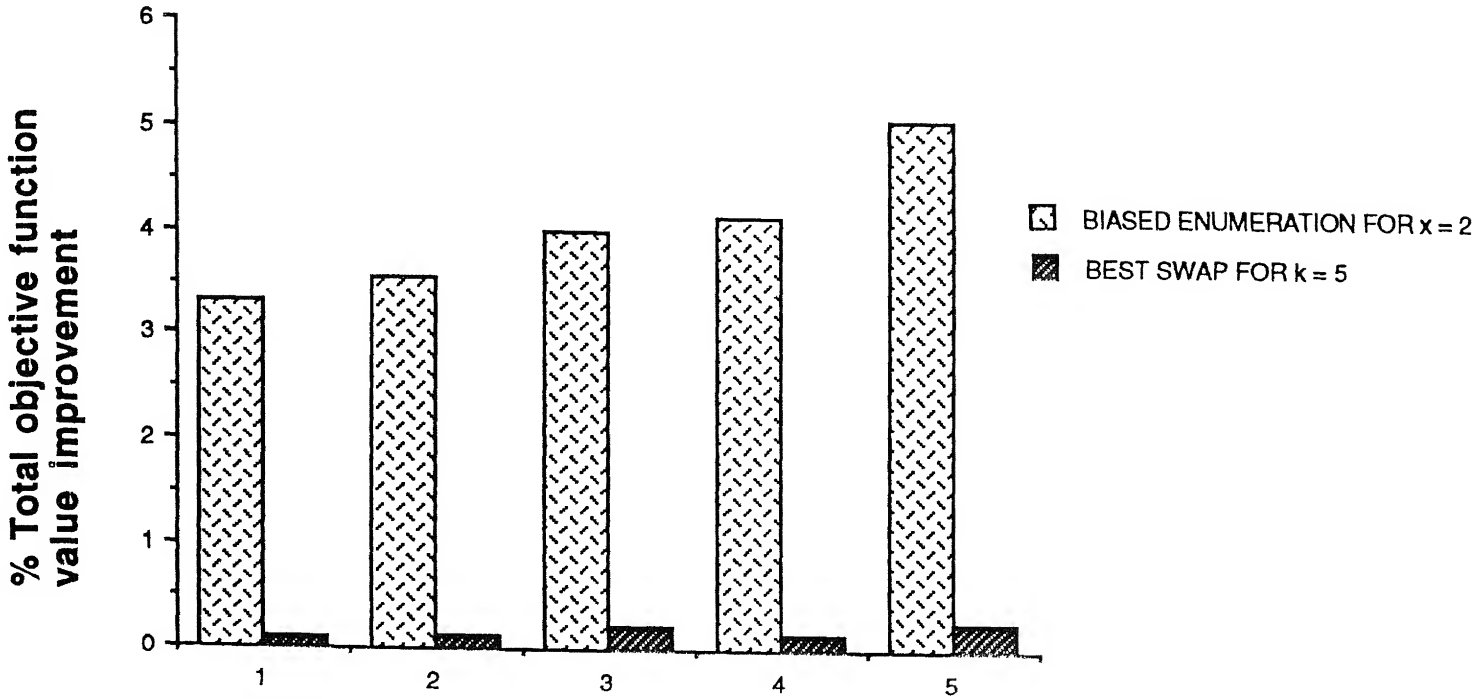


Figure 4.12

Graph showing the comparison between the run time required by swapping strategy and enumeration strategy

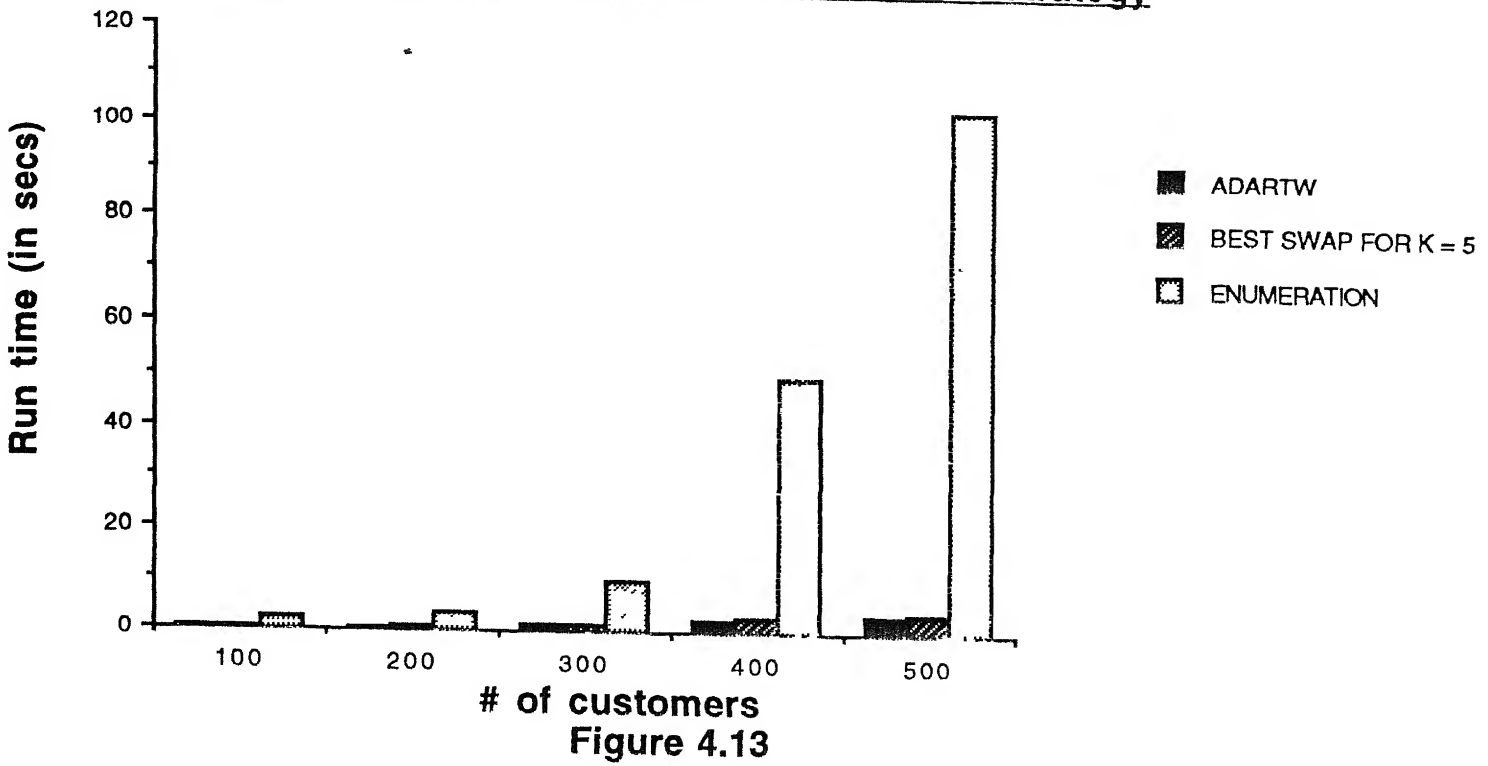


Figure 4.13

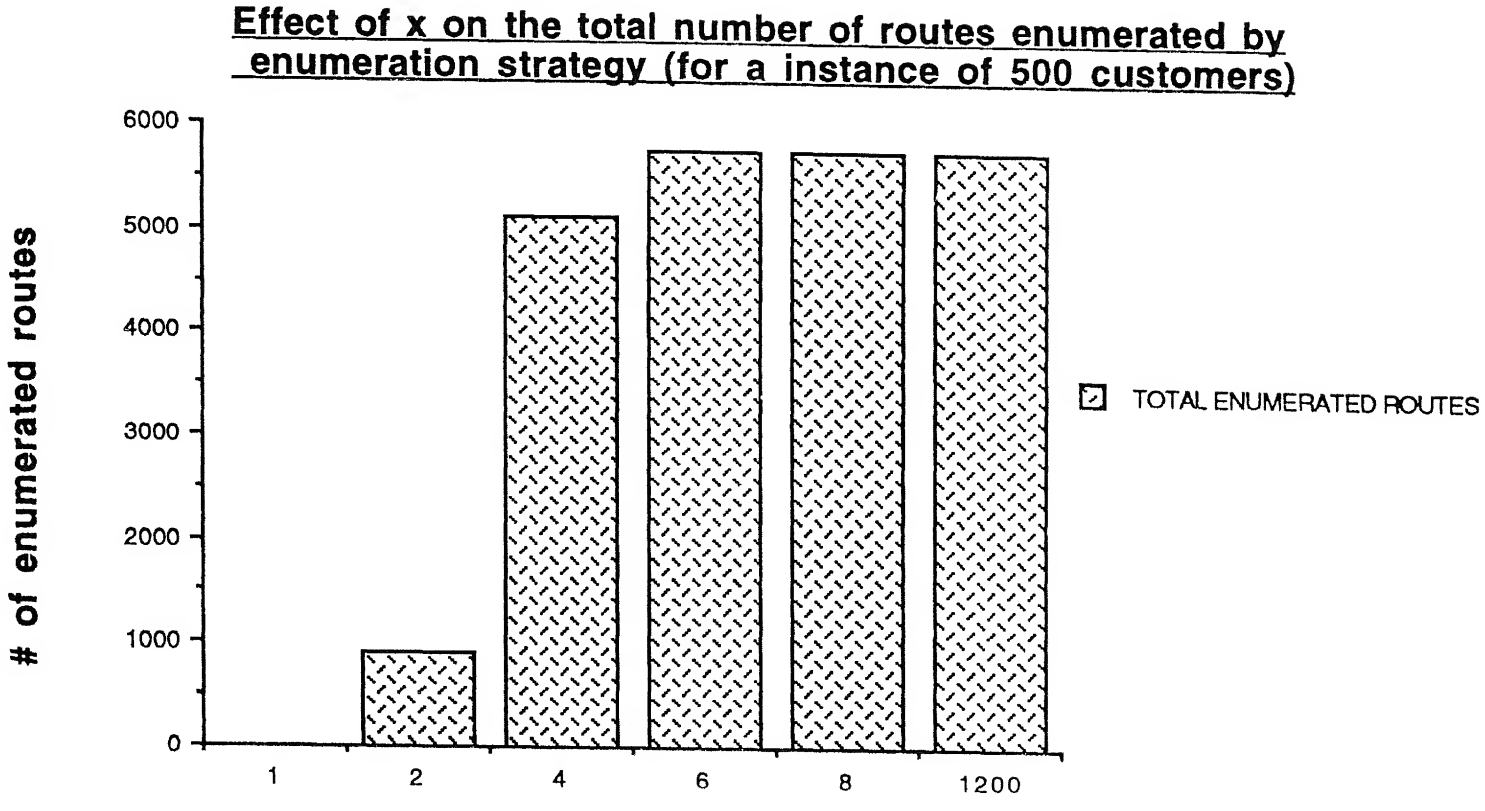

Effect of x on the total number of routes enumerated by enumeration strategy (for a instance of 500 customers)
# of enumerated routes
6000
5000
4000
3000
2000
1000
0
1
2
4
6
8
1200
x
TOTAL ENUMERATED ROUTES

Figure 4.14

Improvements in total objective function value of a vehicle done by enumeration strategy for different value of x (for a instance of 500 customers)

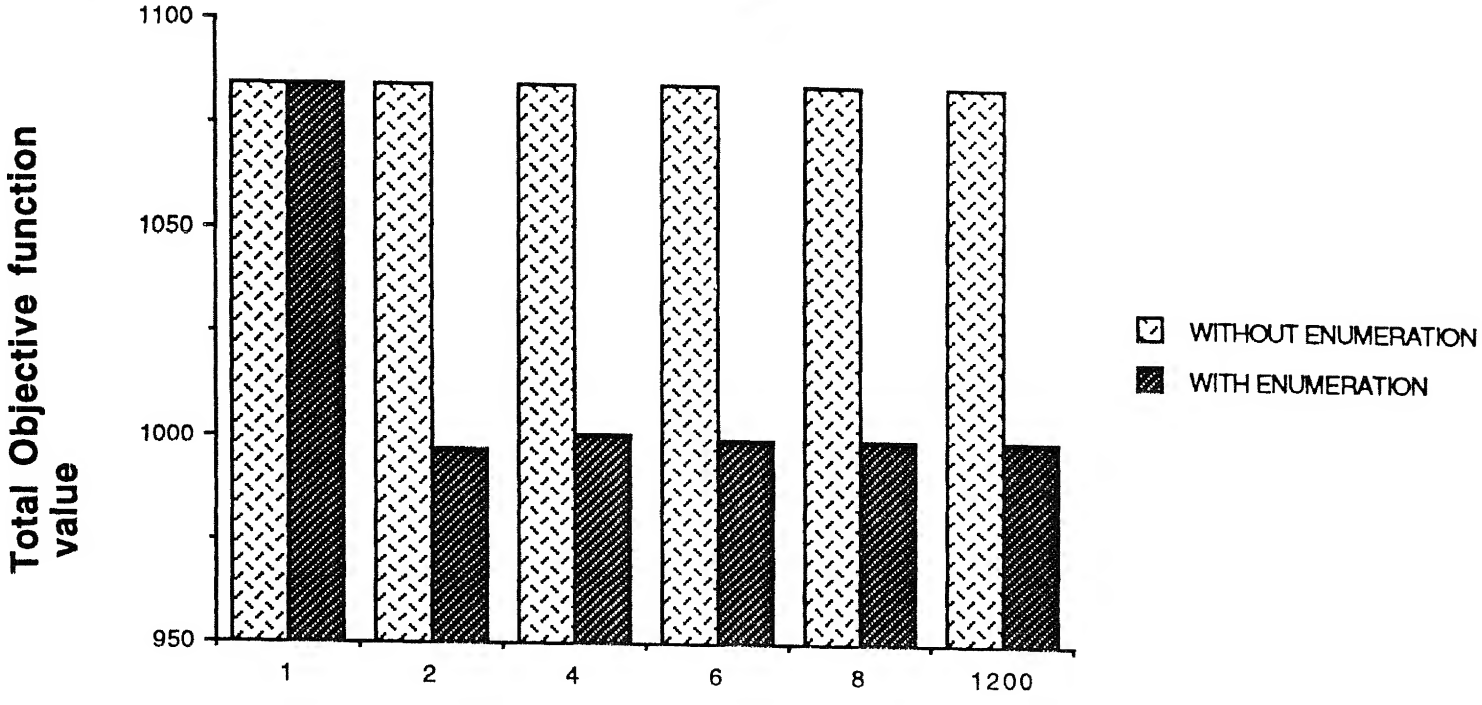


Figure 4.15

highest customer disutility and try to move it. If moving a customer is feasible and this move reduces the overall objective function value of the system only, then we perform this move. Next we choose the customer having second highest customer disutility in the selected vehicle's route and so on After choosing all the customers from the selected vehicle's route in decreasing order of their customer disutility, we choose the next vehicle having second highest service time and so on We keep on doing this until there is any customer left unselected (see Figures A 6 and A 7)

- We choose the first vehicle and choose the customer from the selected vehicle's route having highest disutility, and who has not been chosen so far We try to move this customer if it is possible and profitable. Next, we choose the second vehicle, if this vehicle has any customer who has not been chosen so far, then choose it, otherwise choose next vehicle, and so on After selecting last vehicle, we choose first vehicle until any vehicles is having a customer who is never selected for a move. (See Figure A.8).

- We choose the highest disutility customer, and try to move it if it is possible and profitable, after that choose the customer having the second highest disutility and so on. Keep on doing this until there is any unconsidered customer left for move. (See Figure A.9).

- This strategy is the extension of strategy three. In this, we first run the algorithm described above, and check whether the new solution is better than previous solution or not. If it is not better, then we stop, otherwise we run the algorithm again. We keep on doing this until the new solution is an improved solution (see Figure A.10).

We can make the route feasible after deleting a node from the route with the help of procedure described in Figure A.11. If we are able to find a feasible route only then we can move the selected customer.

Computational results

Figure 4.16 shows that in some cases the first strategy is good, and in some cases second strategy is good, and in some cases third strategy is good. This is because of the different sequence of customer's move. The move of one customer in most of the cases eliminates possibility of many other customers' move So selecting the customer to move is a critical part of the algorithm. We observe that the third strategy is more robust than strategy first and second, which in turn reflects that selecting always highest disutility customer is more appropriate. The reason for this is that there are more improvement chances, if you are able to move the highest disutility customer For a customer having small disutility the chances of improvement are very less, and if you choose this customer, and move it then it will eliminate the possibility of moving the highest disutility customer Sometimes moving customer of less disutility improves more because the disutility improvement of the moving customer not only matters, what matters is the all customers disutility improvement due to the move. So sometime strategy one and sometime strategy two gives good results.

The above paragraph shows the comparison between first, second and third strategy. But Figures 4.17 and 4.18 shows that strategy four is always better than all other three strategies. This is because of its repetitive selection of customers for movement We take the results to see the effect of problem size on improvements as shown in Figures 4.19 and 4.20. We observe that percentage improvement remain approximately the same.

All these strategies use the same algorithm to find the feasible ways of inserting a customer so that the run time required to run the strategy first, second and third are same, and it is double as of ADARTW as shown in Figure 4.21 Strategy four selects customers approximately three times more

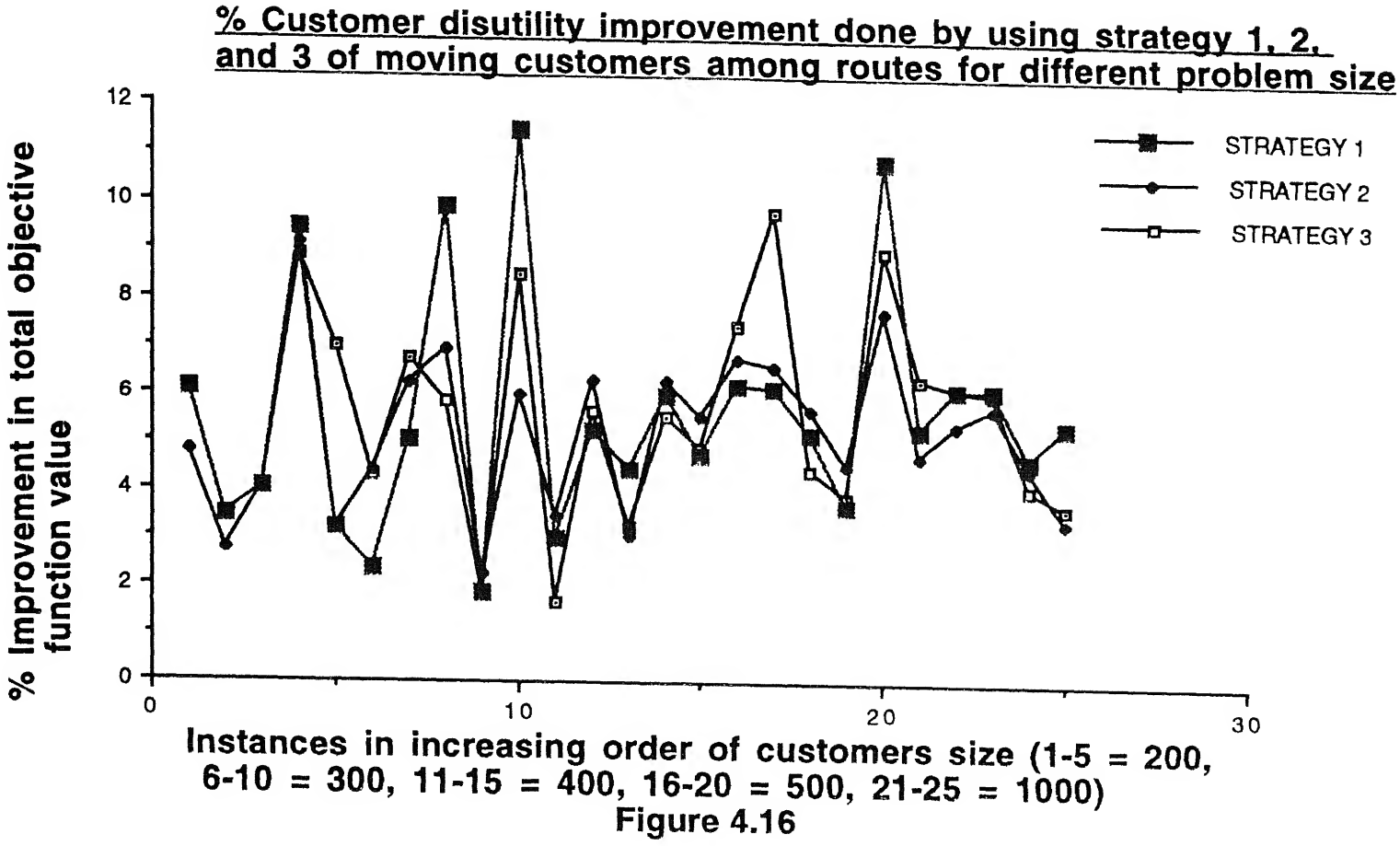

% Customer disutility improvement done by using strategy 1, 2, and 3 of moving customers among routes for different problem size
% Improvement in total objective function value
12
10
8
6
4
2
0
0
10
20
30
STRATEGY 1
STRATEGY 2
STRATEGY 3
Instances in increasing order of customers size (1-5 = 200, 6-10 = 300, 11-15 = 400, 16-20 = 500, 21-25 = 1000)

Figure 4.16

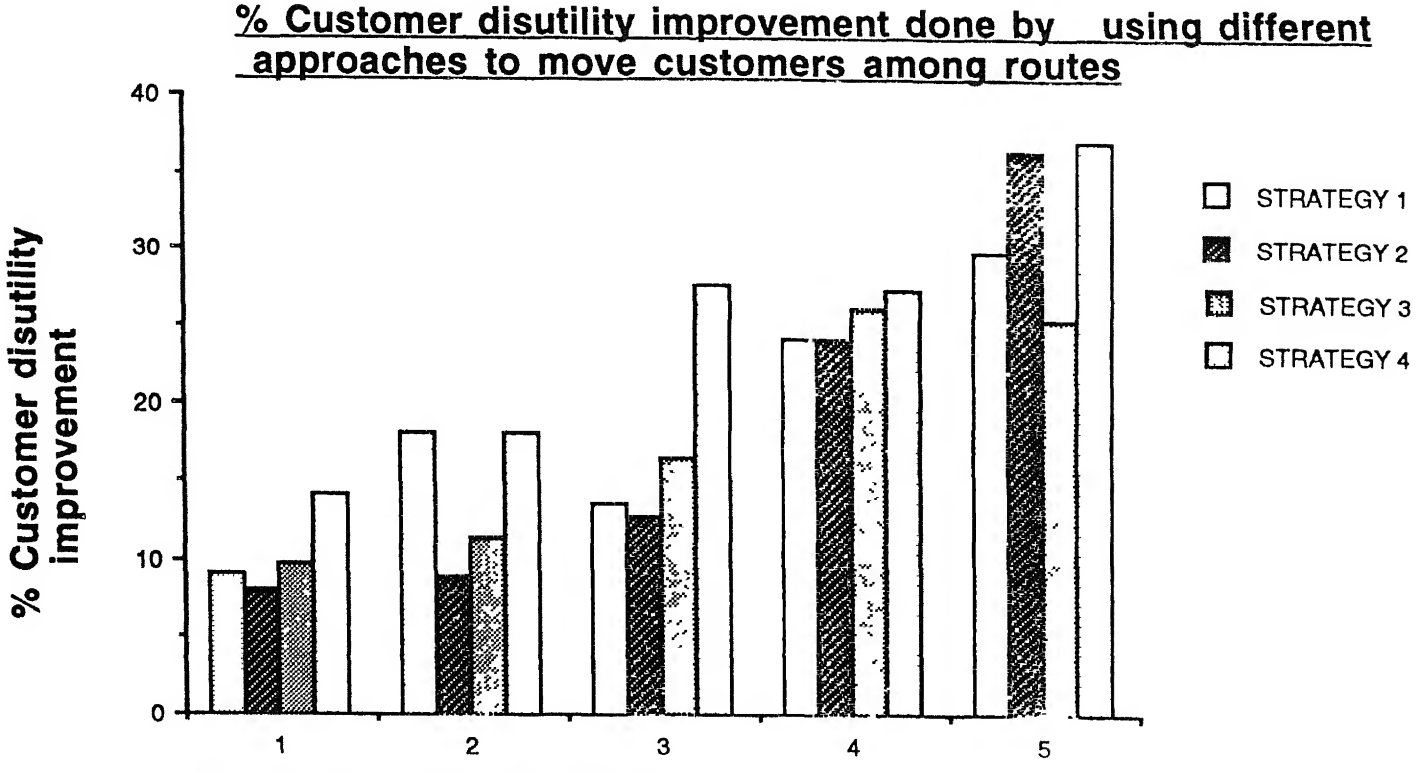

% Customer disutility improvement done by using different approaches to move customers among routes
% Customer disutility improvement
40
30
20
10
0
1
2
3
4
5
STRATEGY 1
STRATEGY 2
STRATEGY 3
STRATEGY 4
Best instances of different customers size
(1 = 1000, 2 = 100, 3 = 100, 4 = 100 , 5 = 100)

Figure 4.17

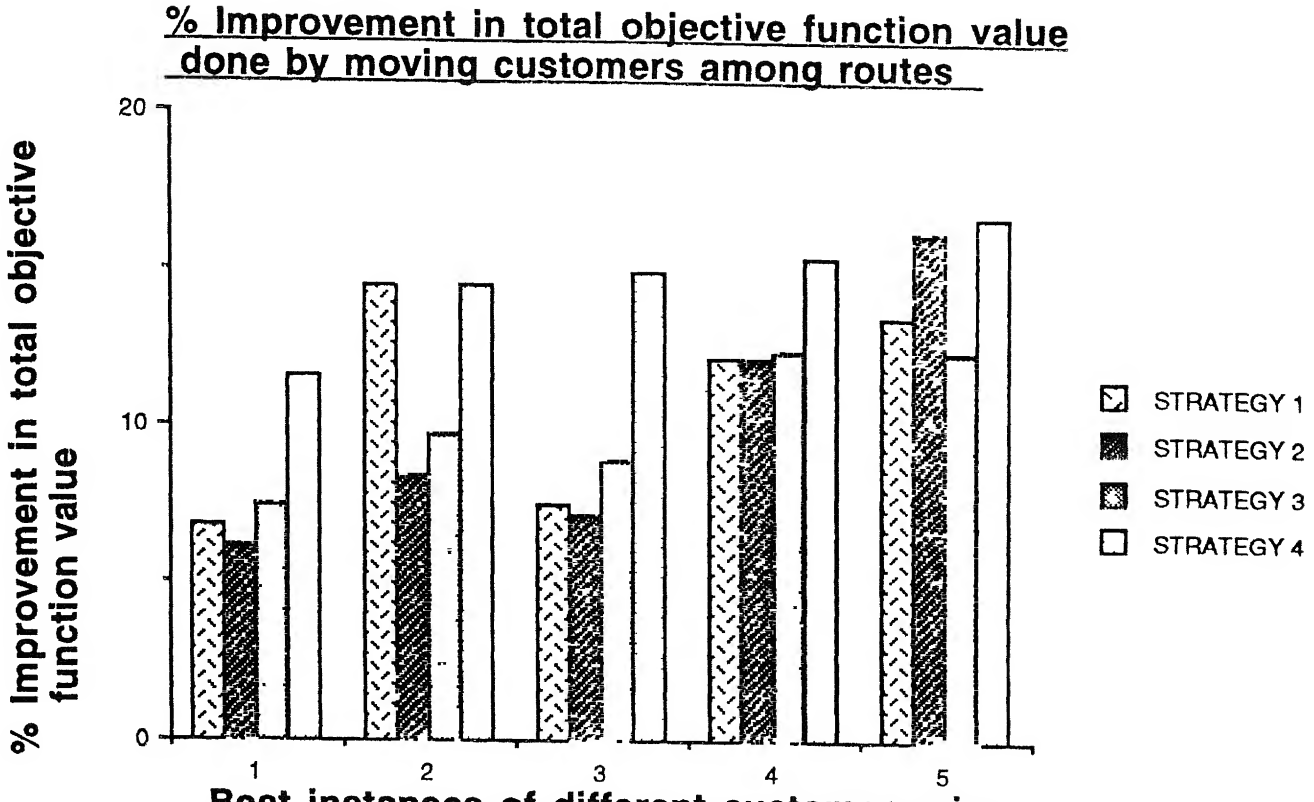

% Improvement in total objective function value done by moving customers among routes
20
10
0
% Improvement in total objective function value
1
2
3
4
5
Best instances of different customers size
(1-1000, 2 = 100, 3 = 100, 4 = 100, 5 = 100)
STRATEGY 1
STRATEGY 2
STRATEGY 3
STRATEGY 4

Figure 4.18

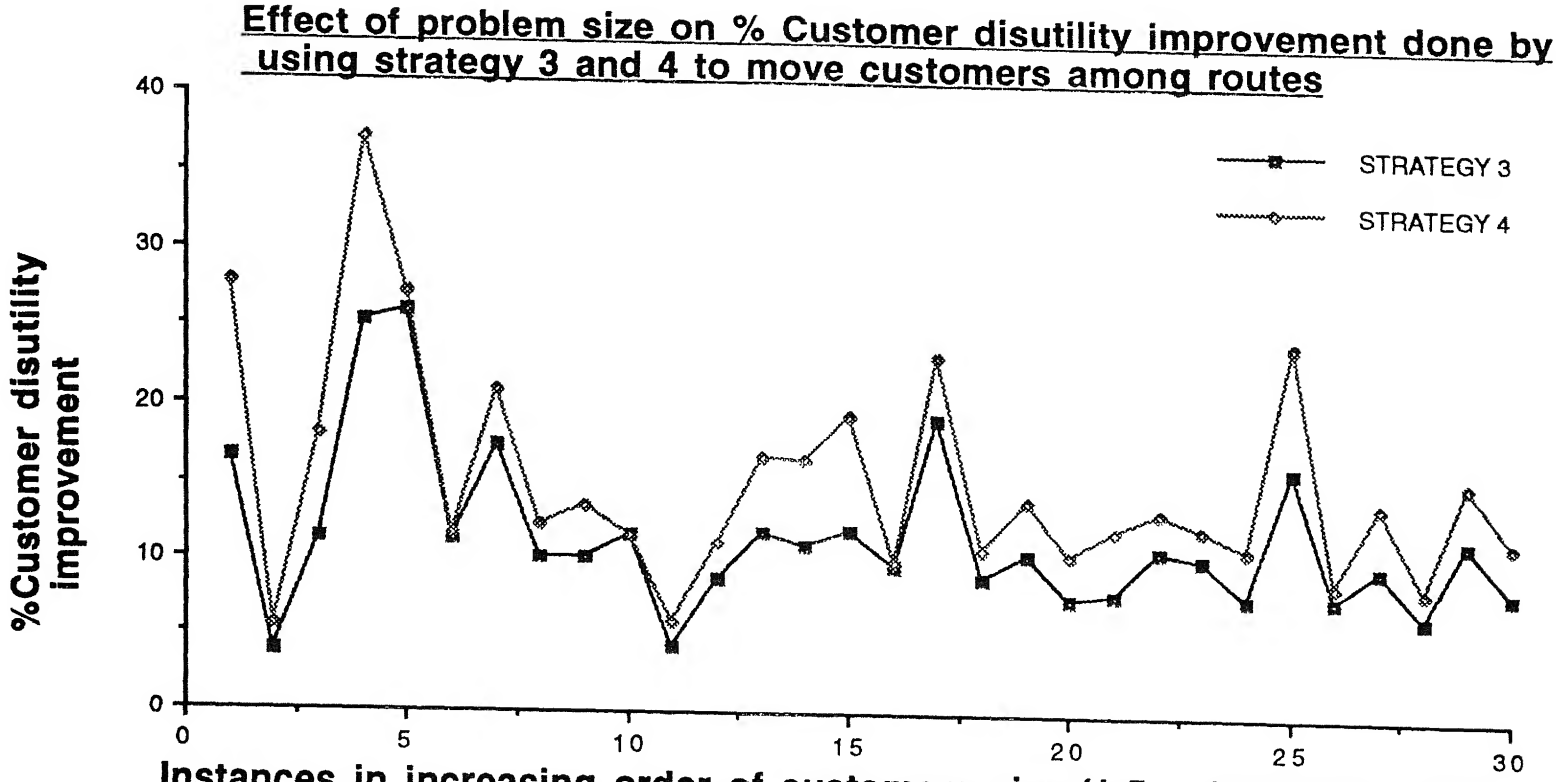

Effect of problem size on % Customer disutility improvement done by using strategy 3 and 4 to move customers among routes
STRATEGY 3
STRATEGY 4
%Customer disutility improvement
0
10
20
30
40
0
5
10
15
20
25
30
Instances in increasing order of customers size (1-5 = 100, 6-10 = 200, 11-15 = 300, 16-20 = 400, 21-25 = 500, 26-30 = 1000)

Figure 4.19

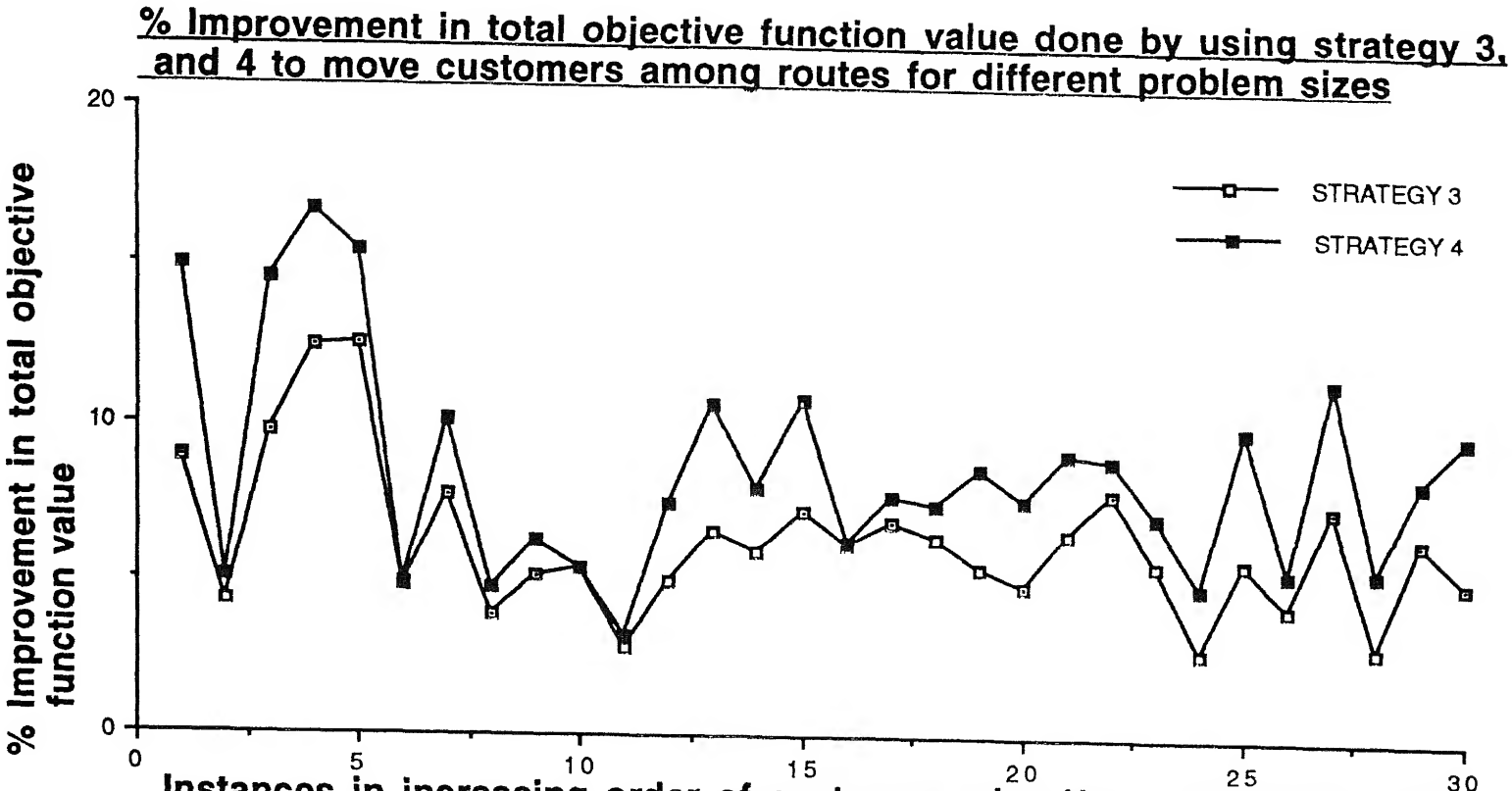

% Improvement in total objective function value done by using strategy 3, and 4 to move customers among routes for different problem sizes
% Improvement in total objective function value
20
10
0
0
5
10
15
20
25
30
STRATEGY 3
STRATEGY 4
Instances in increasing order of customers size (1-5 = 100, 6-10 = 200, 11-15 = 300, 16-20 = 400, 21-25 = 500, 26-30 = 1000)

Figure 4.20

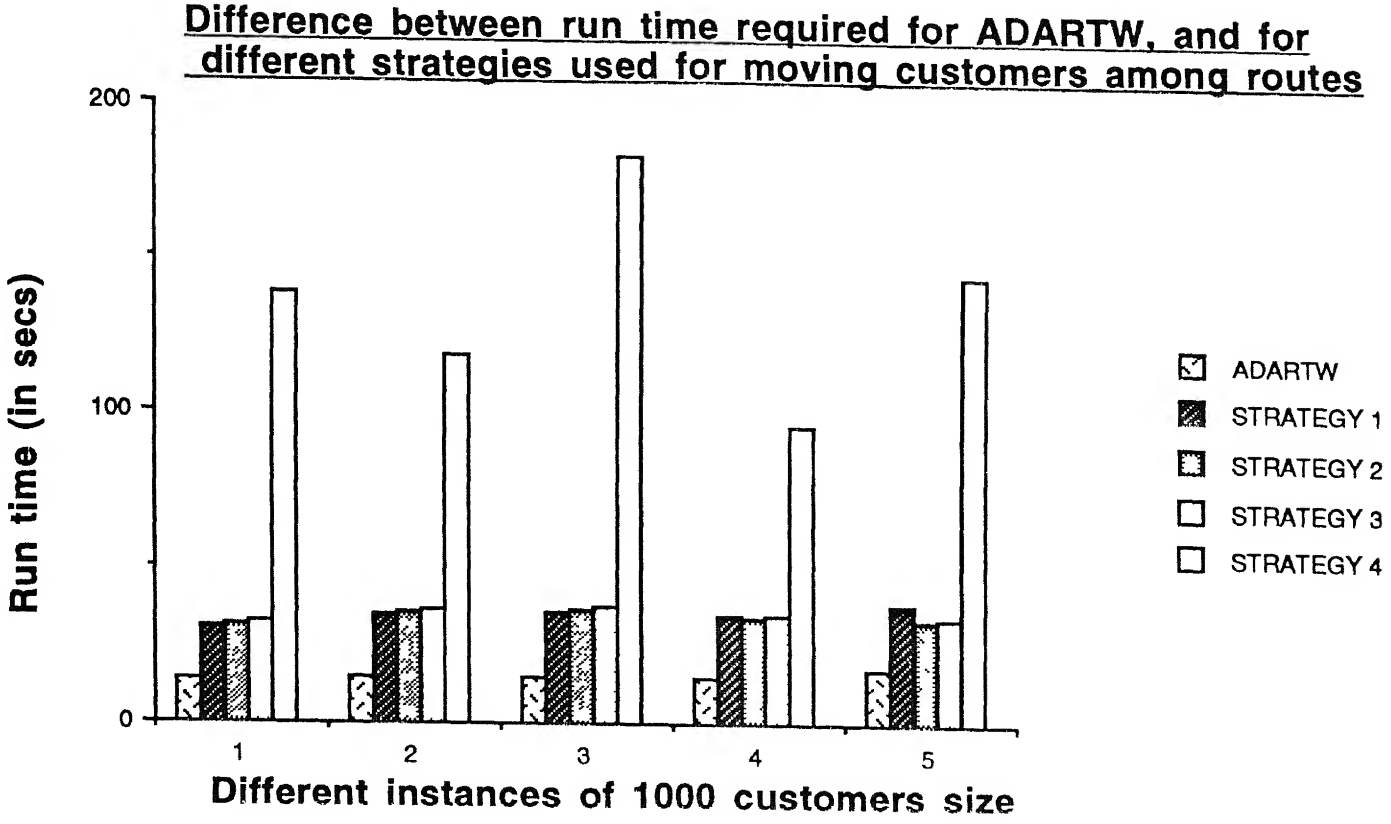

Difference between run time required for ADARTW, and for different strategies used for moving customers among routes
200
100
0
Run time (in secs)
1
2
3
4
5
Different instances of 1000 customers size
ADARTW
STRATEGY 1
STRATEGY 2
STRATEGY 3
STRATEGY 4

Figure 4.21

than strategy one, that is why its run time is approximately three times the strategy first's run time. Figures 4.22 and 4.23 shows the effect of problem size on the run time of the algorithms. We observe that the run time of four strategies always increase in the same proportion as of ADARTW algorithm.

We find this strategy is very successful in dividing the customers equally among the vehicles. Figure 4.24 shows the effect of this strategy on the customer variance. This strategy improves the utilization of the vehicles This is clear from the Figure 4 25

(iv) Two run strategy

This is a global improvement method for advance trip request version.

In ADARTW, we add vehicles to the active fleet size when it is not possible to provide service to a customer with the current fleet size In this approach we run the ADARTW twice. In this approach, we first set fleet size equal to zero and after first run of the algorithm, the fleet size required to serve all customers is known. Now we reduce the fleet size by some percentage y of the known fleet size and run the algorithm again. Figures A.12 and A.13 shows the detailed picture of the algorithm.

Computational results

We take results for different value of y. We observe that the change of y affects the system performance. Figure 4.26 shows the effect of two run strategy on customer disutility for different values of y. Figure 4.27 shows the effect of y value on total objective function value, and Figure 4.28 shows the effect of y value on the system operating cost. We observe that this strategy improves the initial solution for every value of y.

This strategy improves the solution because in this strategy, we always start with some active fleet size, so that the customer initially chosen for insertion

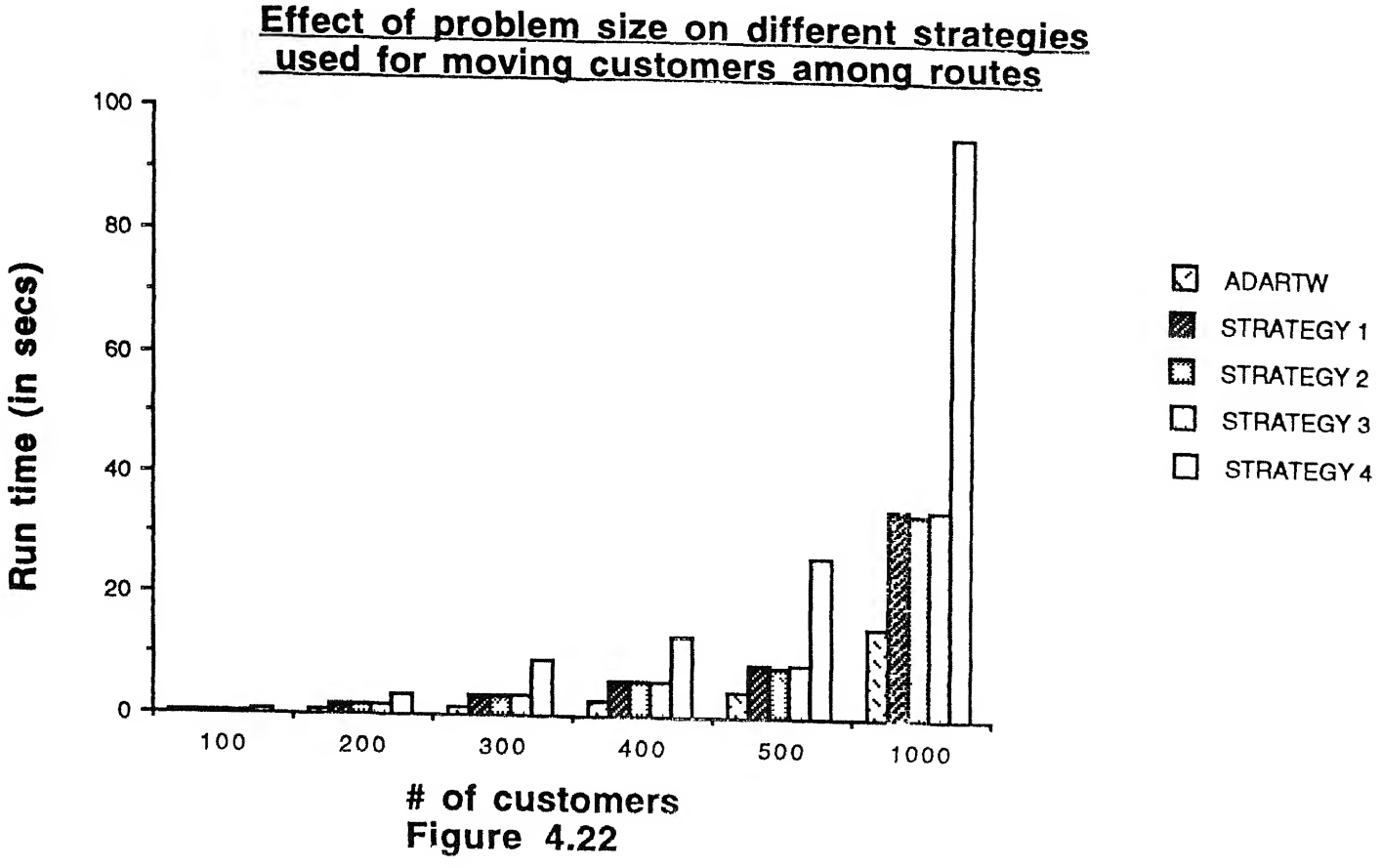

Effect of problem size on different strategies used for moving customers among routes
Run time (in secs)
100
80
60
40
20
0
100
200
300
400
500
1000
# of customers
ADARTW
STRATEGY 1
STRATEGY 2
STRATEGY 3
STRATEGY 4

Figure 4.22

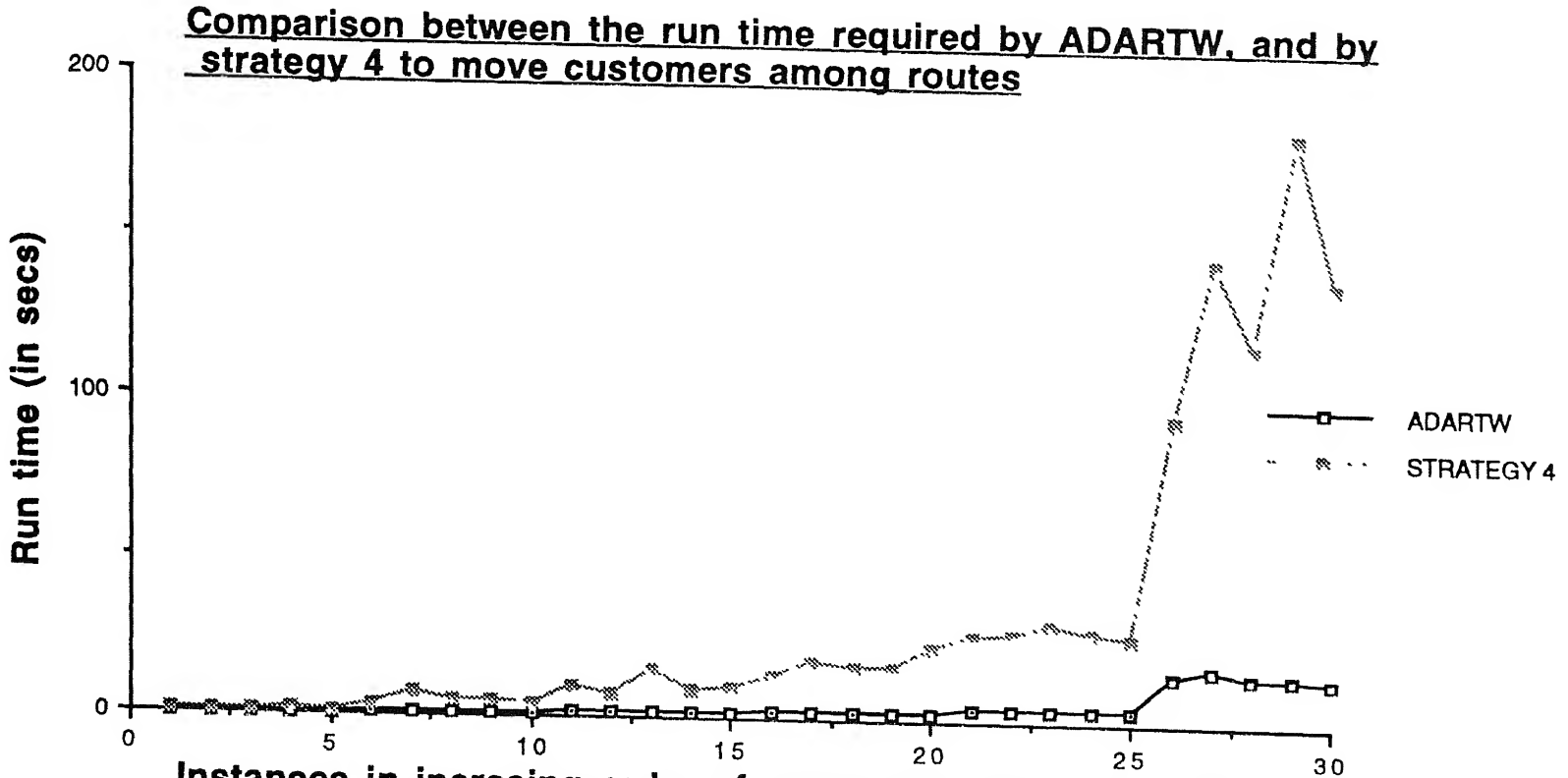

Comparison between the run time required by ADARTW, and by strategy 4 to move customers among routes
Run time (in secs)
200
100
0
0
5
10
15
20
25
30
ADARTW
STRATEGY 4
Instances in incresing order of customers size (1-5 = 100, 6-10 = 200, 11-15 = 300, 16-20 = 400, 21-25 = 500, 26-30 = 1000)

Figure 4.23

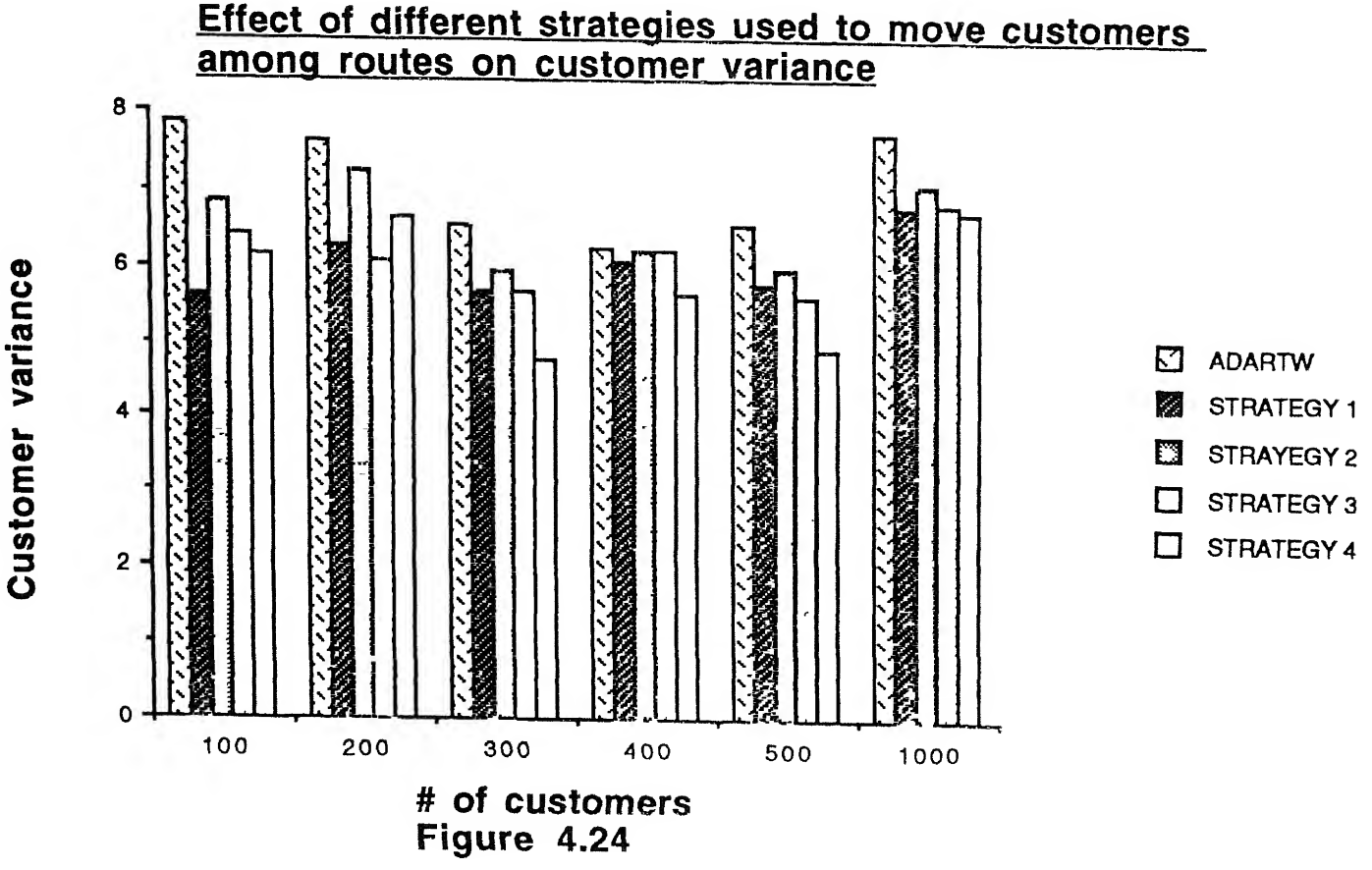

Effect of different strategies used to move customers among routes on customer variance
Customer variance
8
6
4
2
0
100
200
300
400
500
1000
# of customers
ADARTW
STRATEGY 1
STRAYEGY 2
STRATEGY 3
STRATEGY 4

Figure 4.24

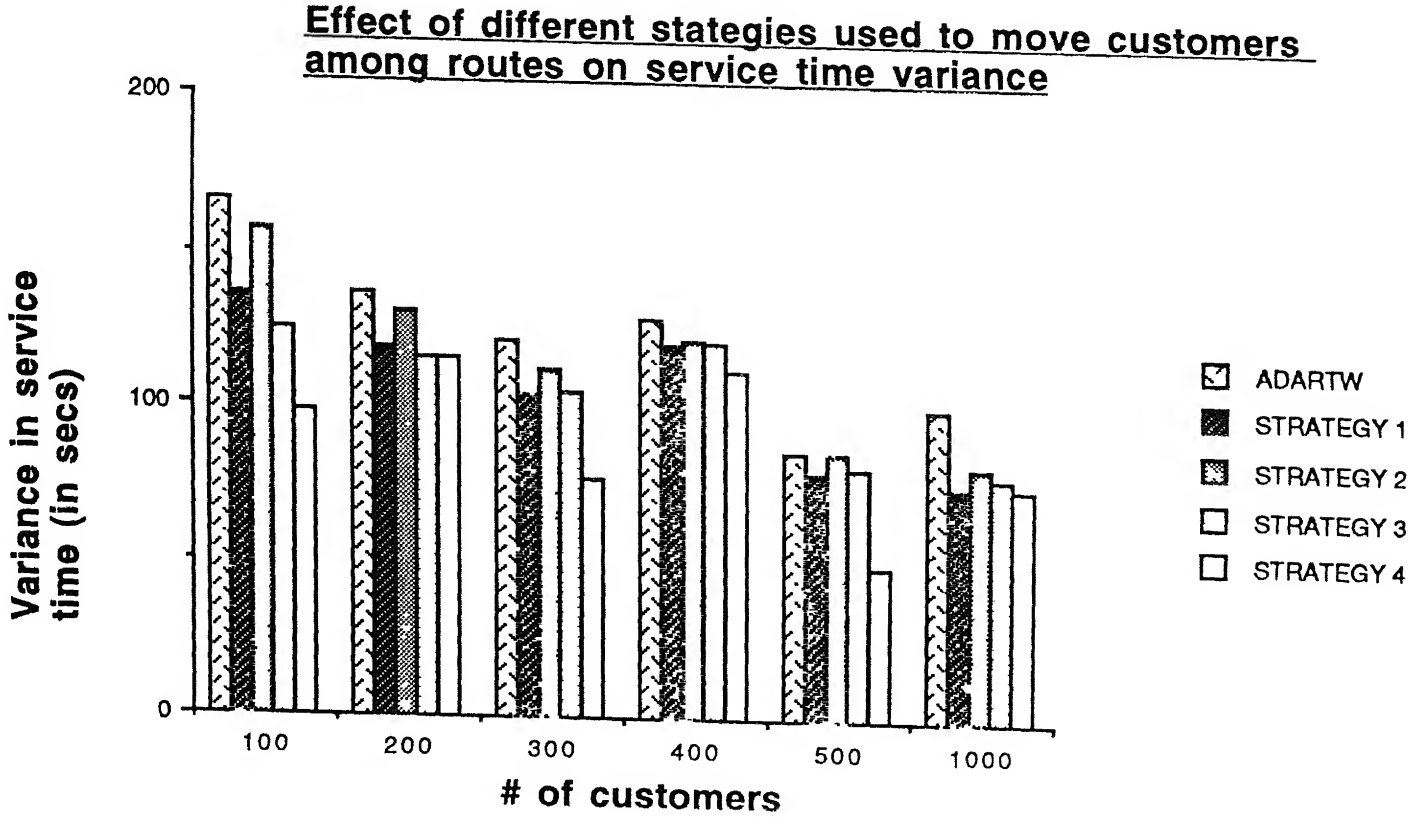

Effect of different stategies used to move customers among routes on service time variance
Variance in service time (in secs)
200
100
0
100
200
300
400
500
1000
# of customers
ADARTW
STRATEGY 1
STRATEGY 2
STRATEGY 3
STRATEGY 4

Figure 4.25

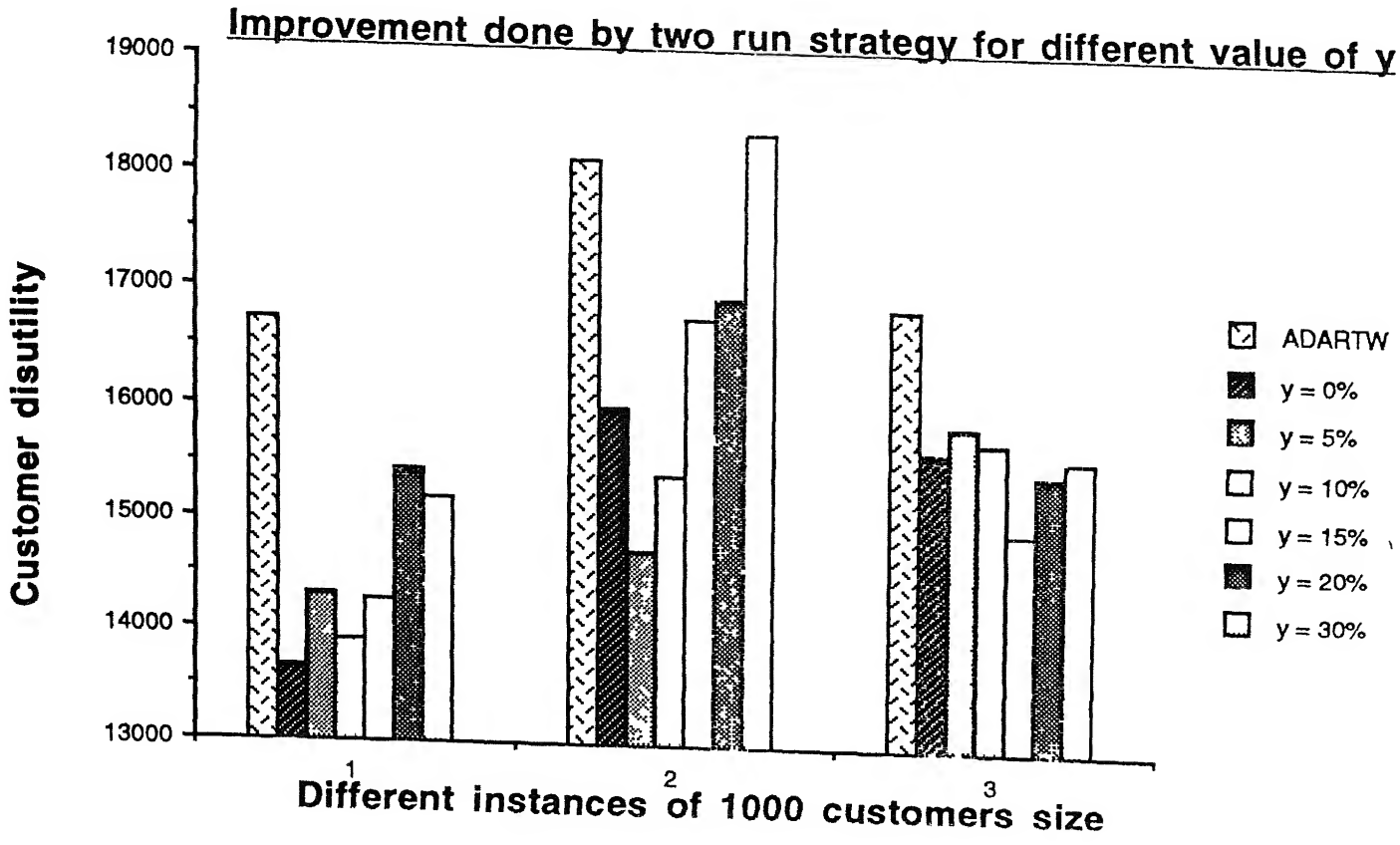

Improvement done by two run strategy for different value of y
Customer disutility
19000
18000
17000
16000
15000
14000
13000
1
2
3
Different instances of 1000 customers size
ADARTW
y = 0%
y = 5%
y = 10%
y = 15%
y = 20%
y = 30%

Figure 4.26

Improvements in total objective function value done by two run strategy for different values of y

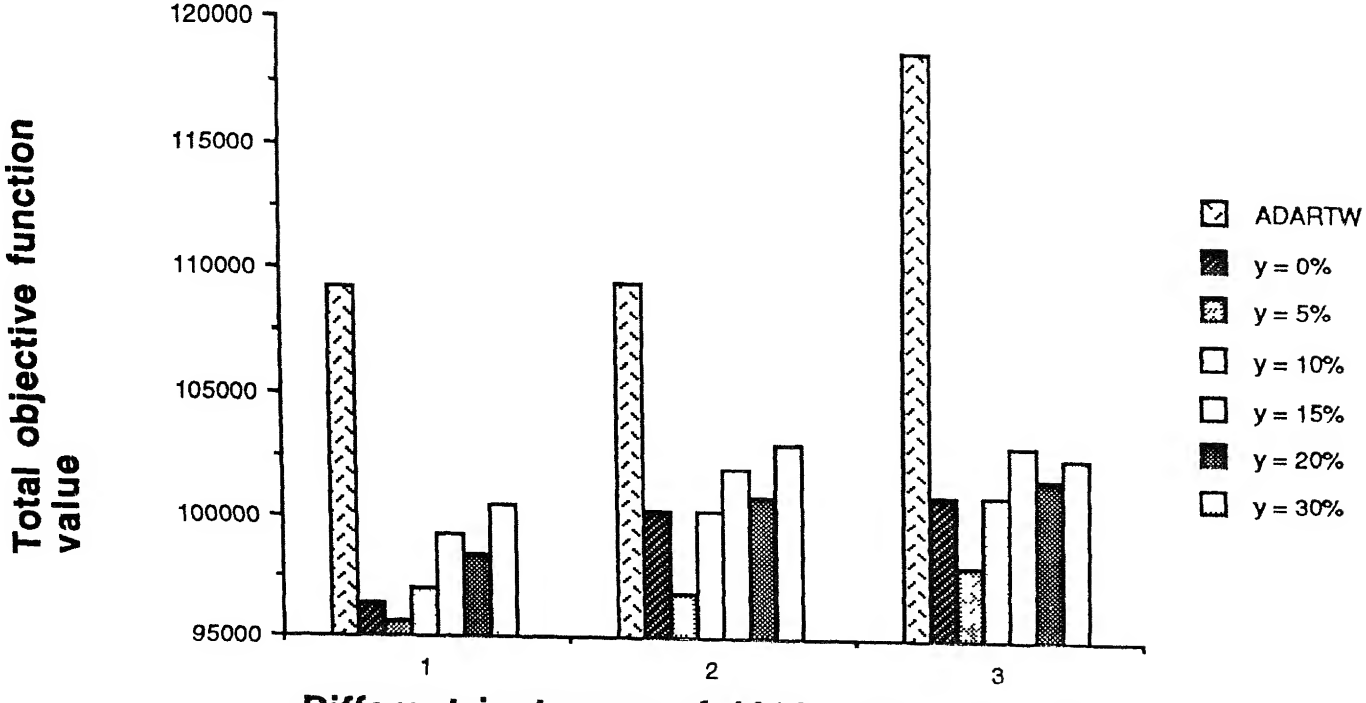


Figure 4.27

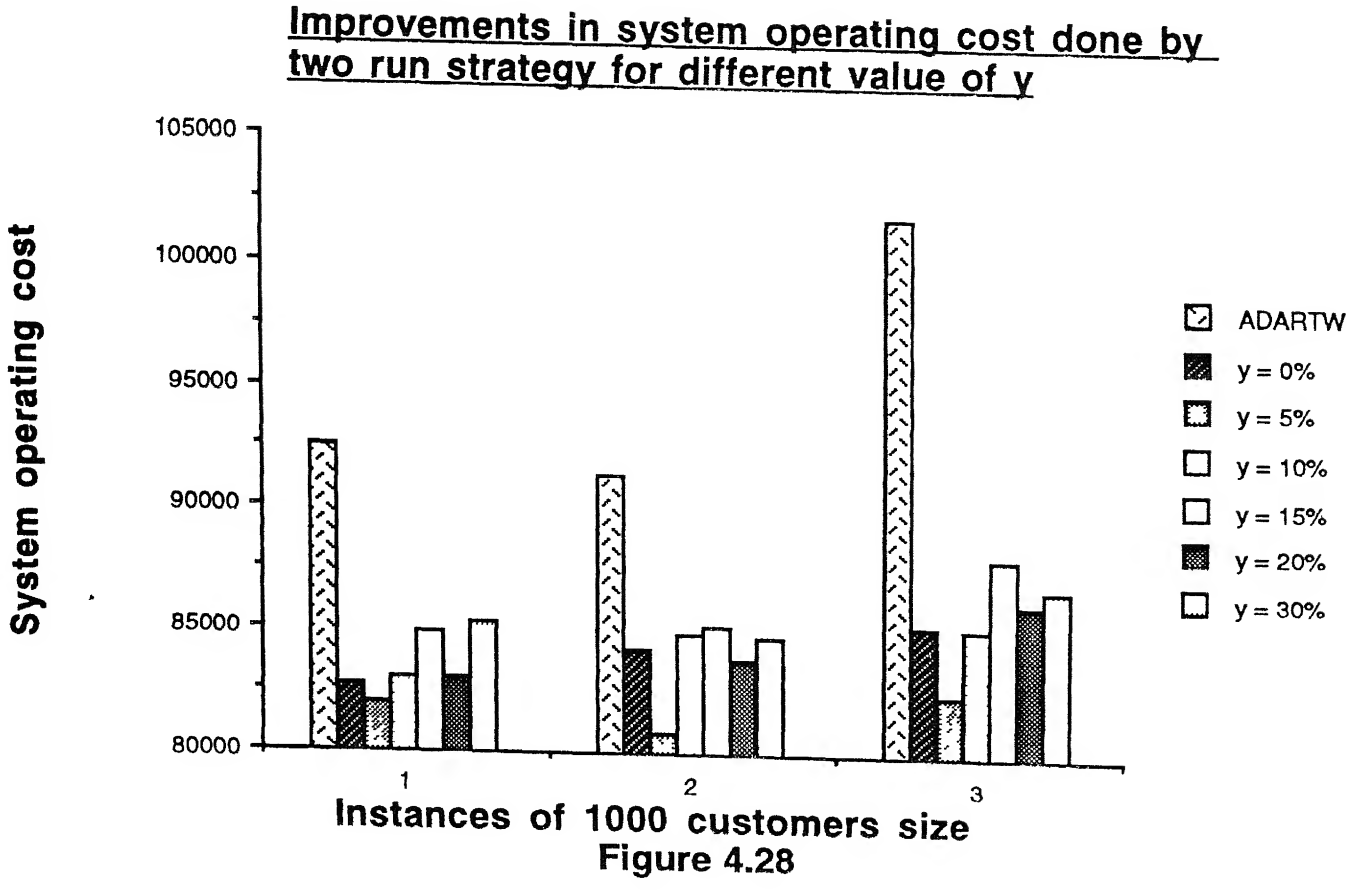

Improvements in system operating cost done by two run strategy for different value of y
System operating cost
105000
100000
95000
90000
85000
80000
1
2
3
Instances of 1000 customers size
ADARTW
y = 0%
y = 5%
y = 10%
y = 15%
y = 20%
y = 30%

Figure 4.28

get opportunities of different routes. This is not the case with ADARTW algorithm In ADARTW algorithms, initially the active fleet size is equal to one, so the customers chosen earlier for insertion do not get option of many routes We observe that y = 5% reduces the total objective function value most, and as we keep on increasing the y value, the improvement performed by the strategy reduces, the same observation we find for the customer disutility Our explanation for such behavior is that as we increase y value the option of many routes for customers reduce that affects the total objective function value. If y = 0% then the improvements are less than the improvement for the value y = 5%. The reason for this we will describe later while discussing the effects of y on total vehicles required. Figures 4.29 and 4 30 show the effect of problem size on the improvements performed by the algorithm.

Figure 4.31 shows the effect of y on total number of vehicles required to serve all customers. This result shows that for y = 5%, total number of vehicles required are the least. The case with y = 5% gives better results than any other value of y. We observe that for y = 0, the total number of vehicles required are always greater than the total vehicles required for y = 5%, that is why, the total objective value for y = 0% is greater than objective function value for y = 5%. For y = 5%, the total number of vehicles required are less, this is because of more options of routes for customers. As we increase the value of y due to the decrease in the total feasible ways of inserting a customer the total number of vehicles required increase.

Two run strategy helps in reducing the customer variance as shown in Figure 4.32. This strategy increases the total service time, which means the vehicles are having smaller slack time. This has also increased the system performance (see Figure 4.33).

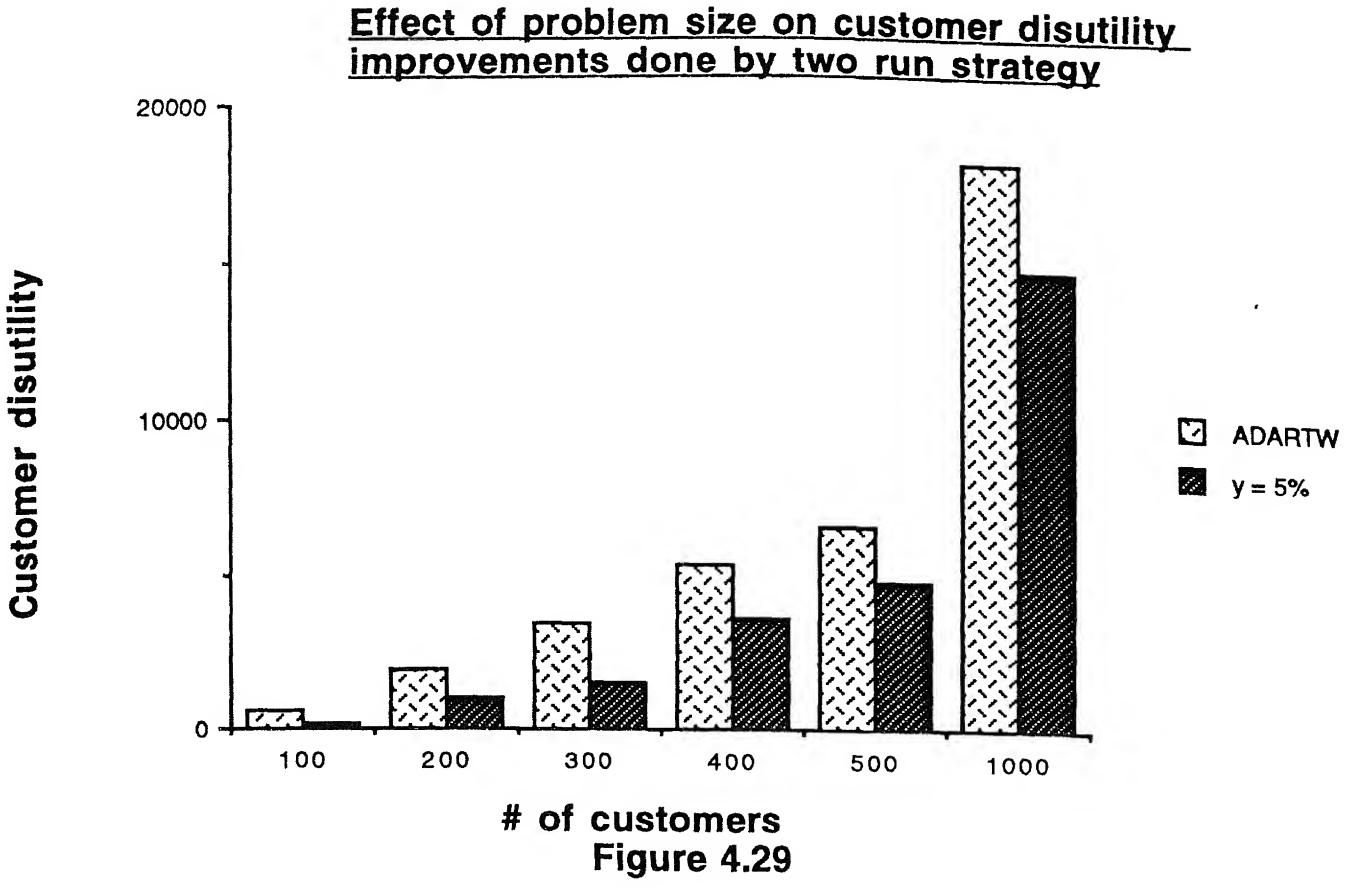

Effect of problem size on customer disutility improvements done by two run strategy
20000
10000
0
Customer disutility
100
200
300
400
500
1000
# of customers
ADARTW
y = 5%

Figure 4.29

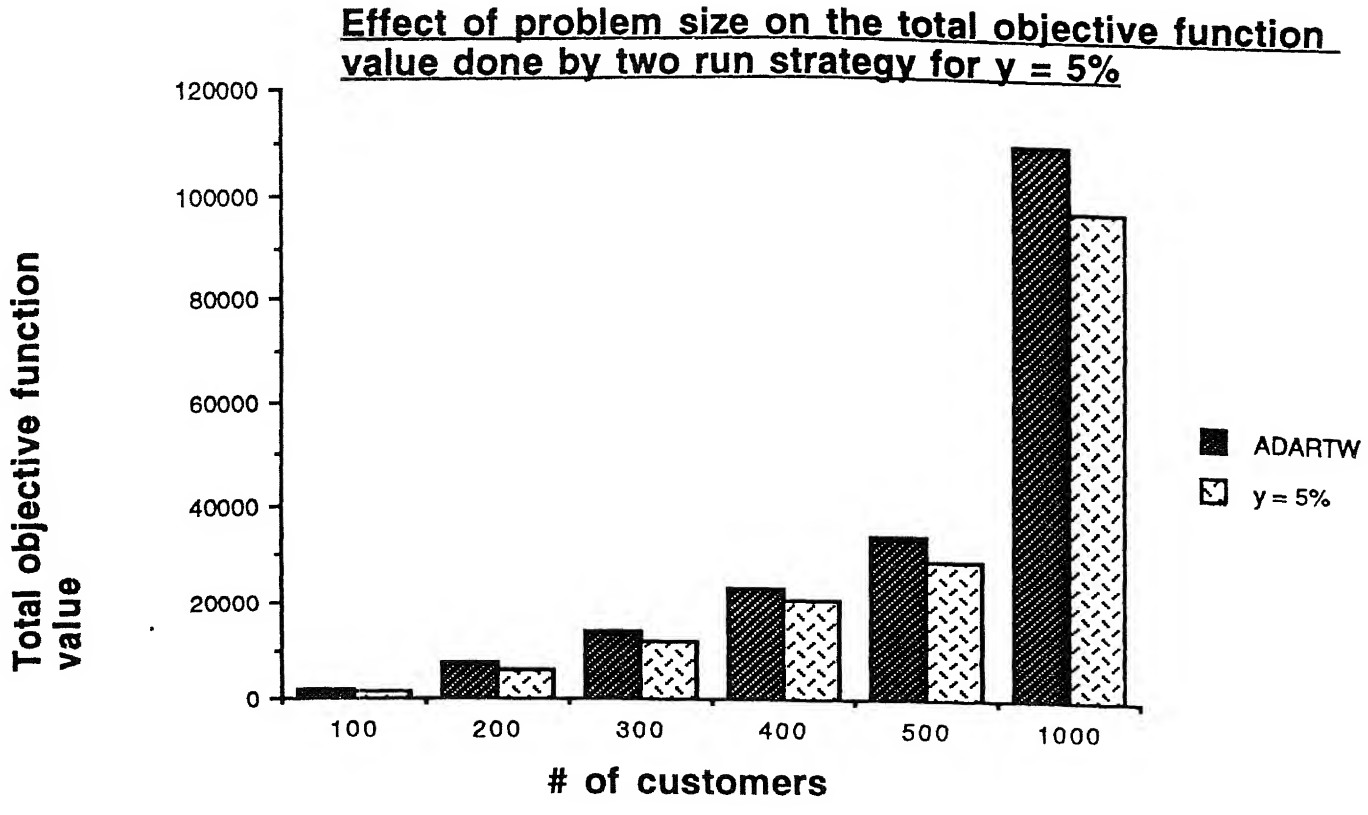

Effect of problem size on the total objective function value done by two run strategy for y = 5%
Total objective function value
120000
100000
80000
60000
40000
20000
0
100
200
300
400
500
1000
# of customers
ADARTW
y = 5%

Figure 4.30

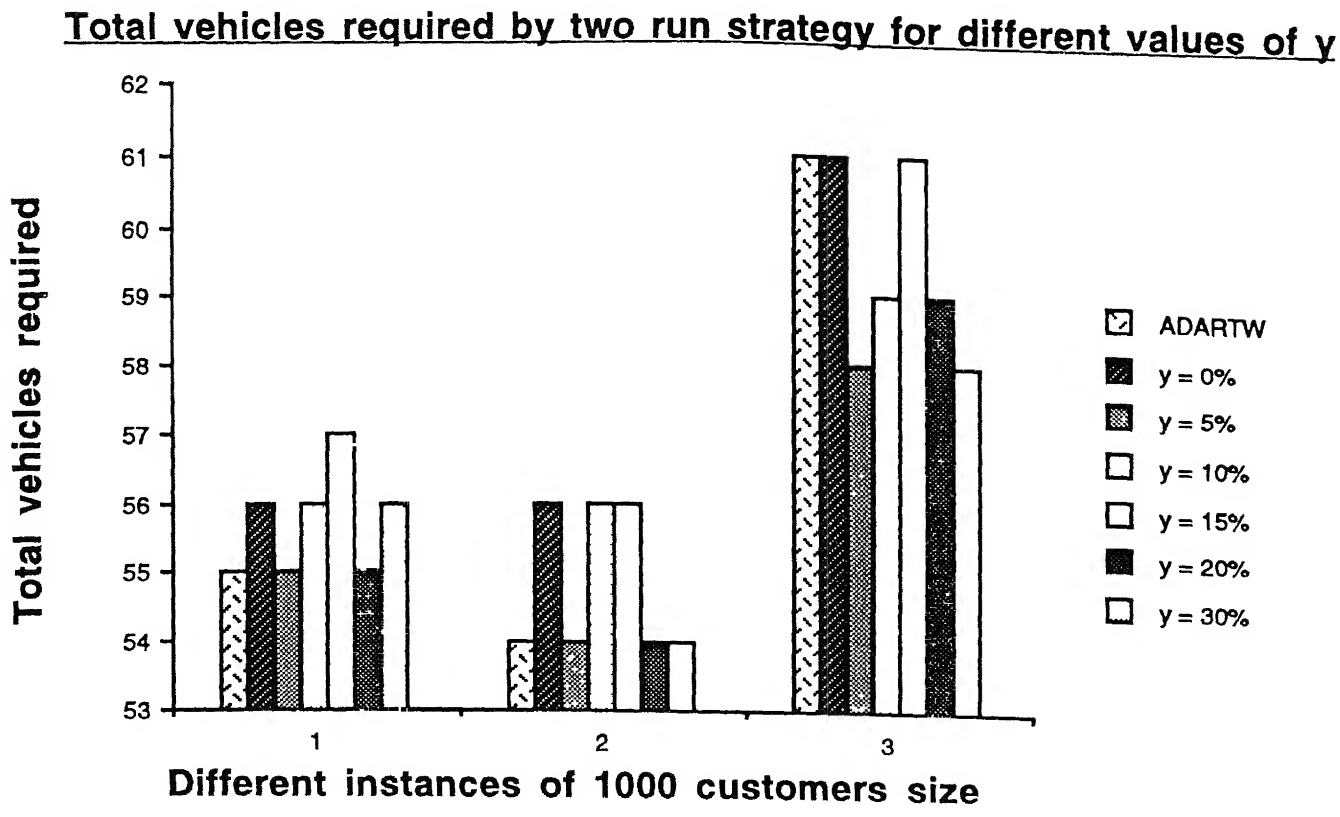

Total vehicles required by two run strategy for different values of y
Total vehicles required
62
61
60
59
58
57
56
55
54
53
1
2
3
Different instances of 1000 customers size
ADARTW
y = 0%
y = 5%
y = 10%
y = 15%
y = 20%
y = 30%

Figure 4.31

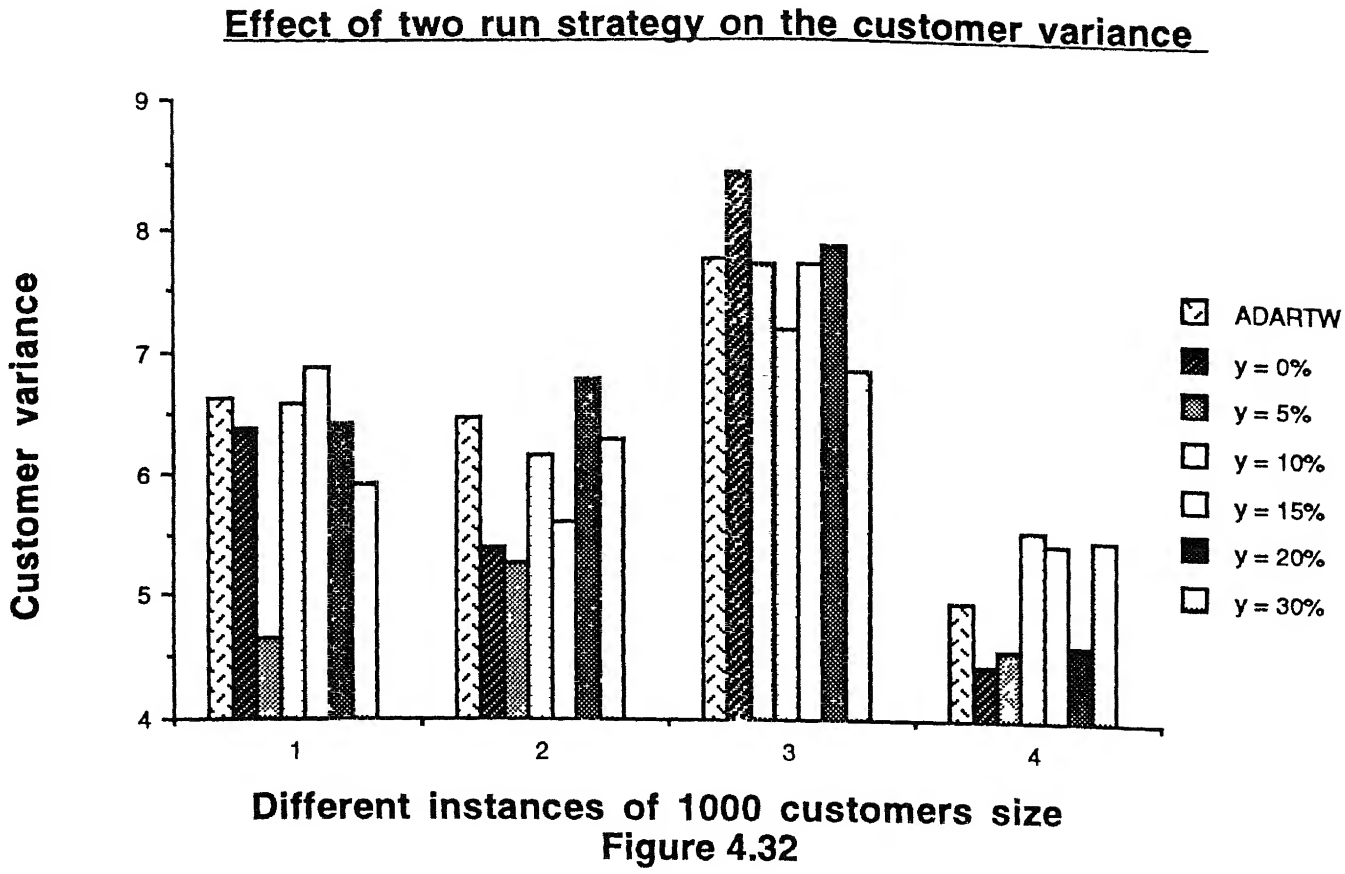

Effect of two run strategy on the customer variance
Customer variance
9
8
7
6
5
4
1
2
3
4
Different instances of 1000 customers size
ADARTW
y = 0%
y = 5%
y = 10%
y = 15%
y = 20%
y = 30%

Figure 4.32

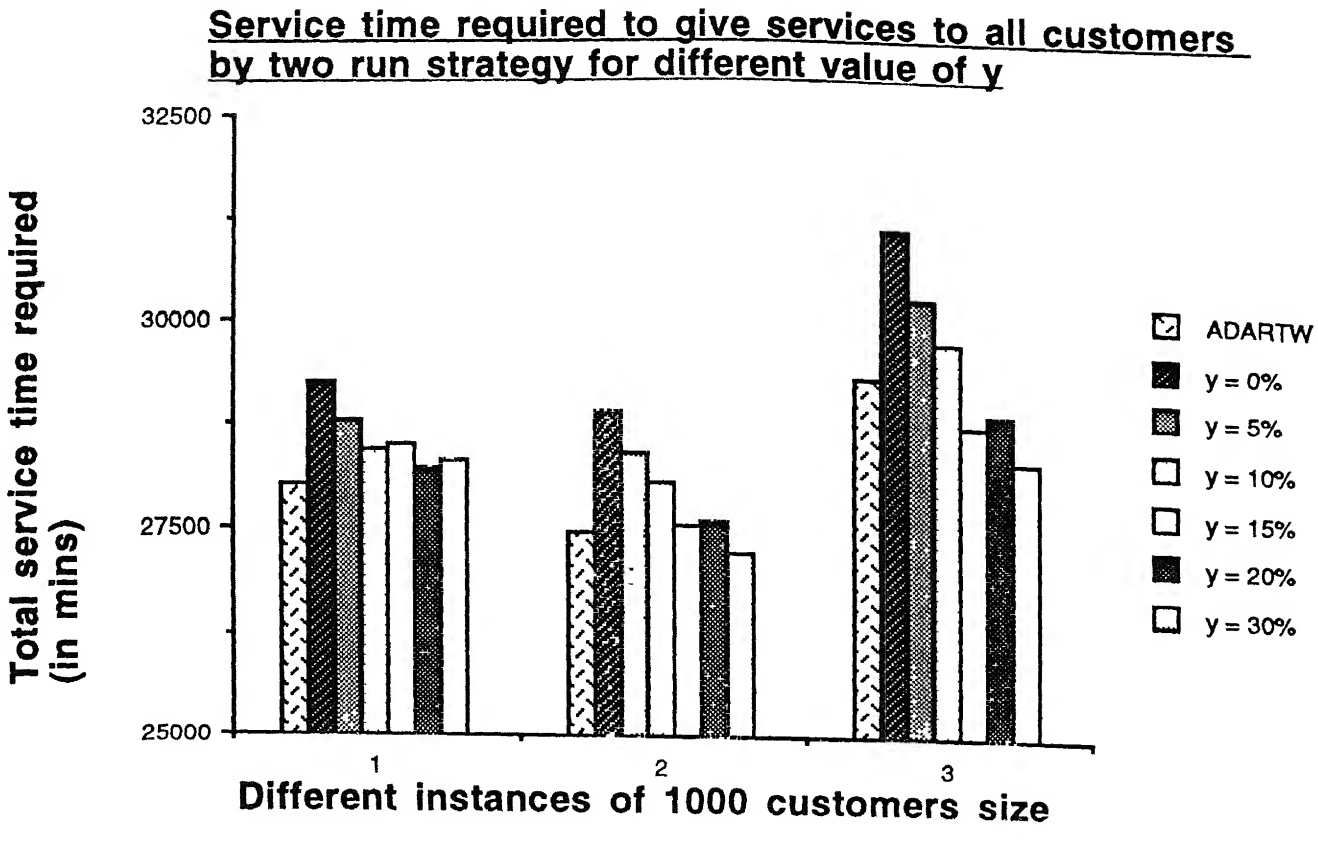

Service time required to give services to all customers by two run strategy for different value of y
Total service time required (in mins)
32500
30000
27500
25000
1
2
3
Different instances of 1000 customers size
ADARTW
y = 0%
y = 5%
y = 10%
y = 15%
y = 20%
y = 30%

Figure 4.33

Figure 4.34 shows that this algorithm takes almost twice the time taken by ADARTW algorithm for all values of y .

4.3 IMMEDIATE REQUEST VERSION

The ADART system anticipates that there will be new trip requests that will be handled by the on board computers of vehicles in an autonomous fashion (i.e., without the aid of a central computer). So we develop a simulation model that is capable of handling both advance requests and immediate requests. We use insertion based heuristics to handle immediate requests. Figure A.14 shows the detailed picture of the heuristic.

We use two strategies for handling immediate requests based on, at what time the customer should be considered for insertion. The two strategies are as follows :

- We try to handle immediate requests **b** minutes before the new trip request's earliest pickup time.
- We try to handle immediate requests as soon as it will come but at least **b** minutes before (according to immediate request arrival time).

Vehicles require some time to go from depot to customer's origin so the immediate requests must be conveyed to the ADART system a fixed time before their earliest pickup time. The minimum difference between EPT and arrival time of new trip request is the parameter **b** of our algorithm which we can change. The vehicle's computer runs the program at discrete intervals of time, which we have taken to be fifteen minutes. This will decide how frequently the on-board vehicle's computer runs the algorithm. For new trip requests, it is not possible to find U values, because due to the absence of

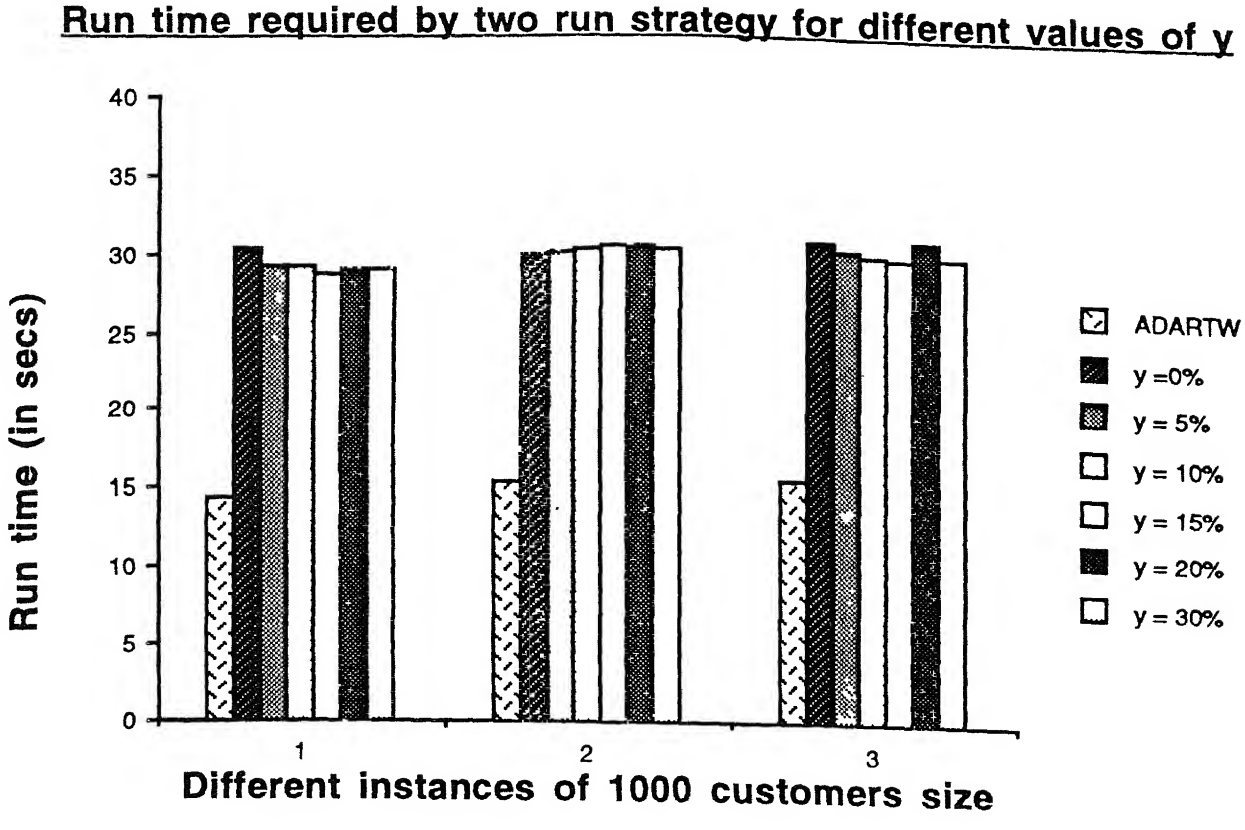

Run time required by two run strategy for different values of y
Run time (in secs)
40
35
30
25
20
15
10
5
0
1
2
3
Different instances of 1000 customers size
ADARTW
y =0%
y = 5%
y = 10%
y = 15%
y = 20%
y = 30%

Figure 4.34

centralized computer it is difficult to find the number of system users and the number of effective vehicles corresponding to a new trip request so we take it 0 1 for all new trip request. This parameter is also changeable.

As the time progresses the vehicle serves more and more customers. For those customers those have been picked and are yet on board, we set their pickup node's ET = LT = AT so that further insertion of customers does not disturb the timings of this node. We set the ET = LT = AT also for the node for which the vehicle is moving. Because vehicle is already moving for that station, we will avoid changing the vehicle's arrival time at that station.

We incorporate local improvement method to get better solutions. By restricting the nodes which we can swap, we can improve the result with the help of best swap algorithm described earlier. The nodes having ET = LT = AT can not take part in swapping because these are the nodes which either have been served or for which vehicle is moving. Except these nodes, all other nodes can take part in swapping.

We design an enumeration scheme to handle immediate requests. Figures A 15 and A.16 shows a clear picture of this scheme.

In this scheme, we try to find the enumerated route for a subset of customers plus new trip request. In this approach the route is not made empty initially. First in the route, we locate a node at which vehicle is empty, and vehicle's arrival time on that node should be greater than current time. The customers after this node and the new trip request take part in enumeration.

Computational results

We take results to see the effect of **b** on each strategy. For strategy one, Figure 4.35 shows that as **b** increases, the customer disutility decreases. Figure 4.36 shows the effect of b on total objective function value. Our explanation for

Effect of b on customer disutility obtained (for strategy 1 of handling new trip request)

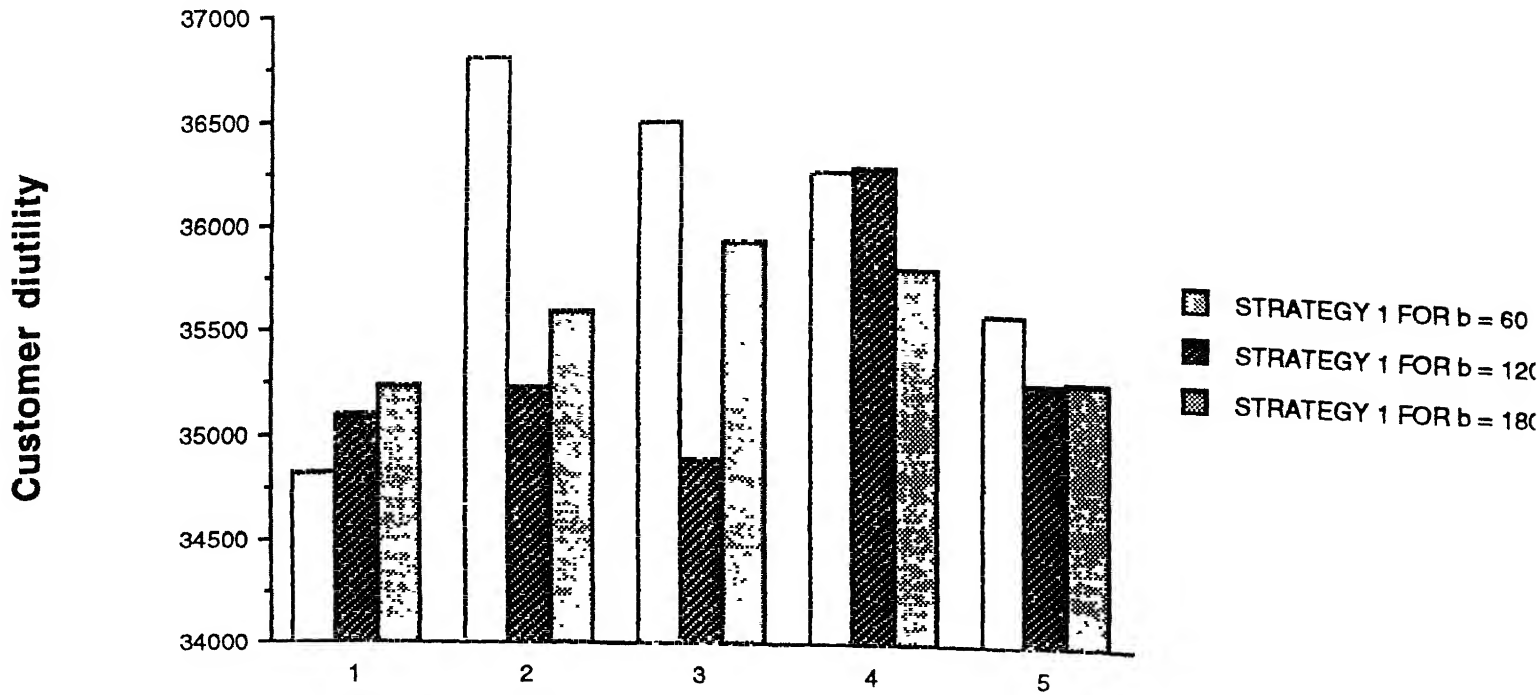


Figure 4.35

Effect of b on total objective function value (for strategy 1 of handling new trip request)

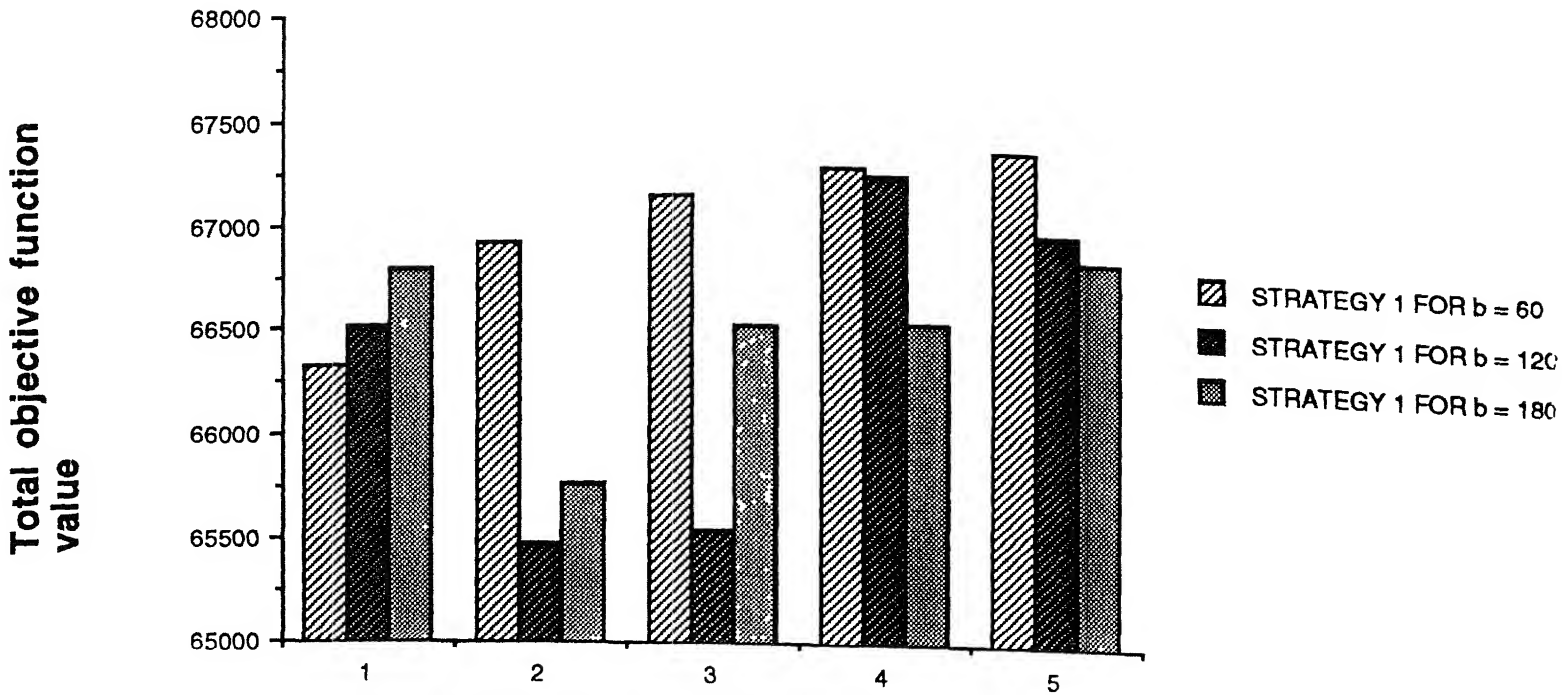


Figure 4.36

such behavior is that as we increase the value of the feasible ways of inserting a customer increase that increase the system performance. Same is the case with strategy 2. Figures 4 37 and 4.38 shows the effect of **b** on the customer disutility and the total objective function value when we use the second strategy. When we compare strategy first and second, we find that second strategy is better as shown in Figures 4.39, 4.40, 4 41, 4.42, 4.43 and 4 44. The reason for this is also the same.

We used the swapping strategy to new trip request version to improve the result. Figuresure 4.45 and 4.46 shows the improvements. For improving the results, we used the best swap strategy with **k** = 2 because the results for this strategy are the most attractive.

4.4 CANCELLATIONS

Sometimes customers change their plans. For example, Ashok was planning to go to shopping complex so he requested for ADART service, later he changed his plan. For such cases, ADART provides a facility of cancellation. If the customer will intimate the system that he/she does not want the requested service at least c time before his/her request's EPT. Where c is a parameter which we can change. We assume that customers only do cancellations, if they requested for the service in advance. For a advance request, the time difference between request's EPT and request's arrival time is significant, so there is more likelihood that customer may change his/her plan. We designed two algorithms for cancellation one for the advanced request version and other for new trip request version (See Figuresure A.17 and A.18). In these approaches we used complete enumeration to generate the route for a vehicle having a route without canceled requests. (We use

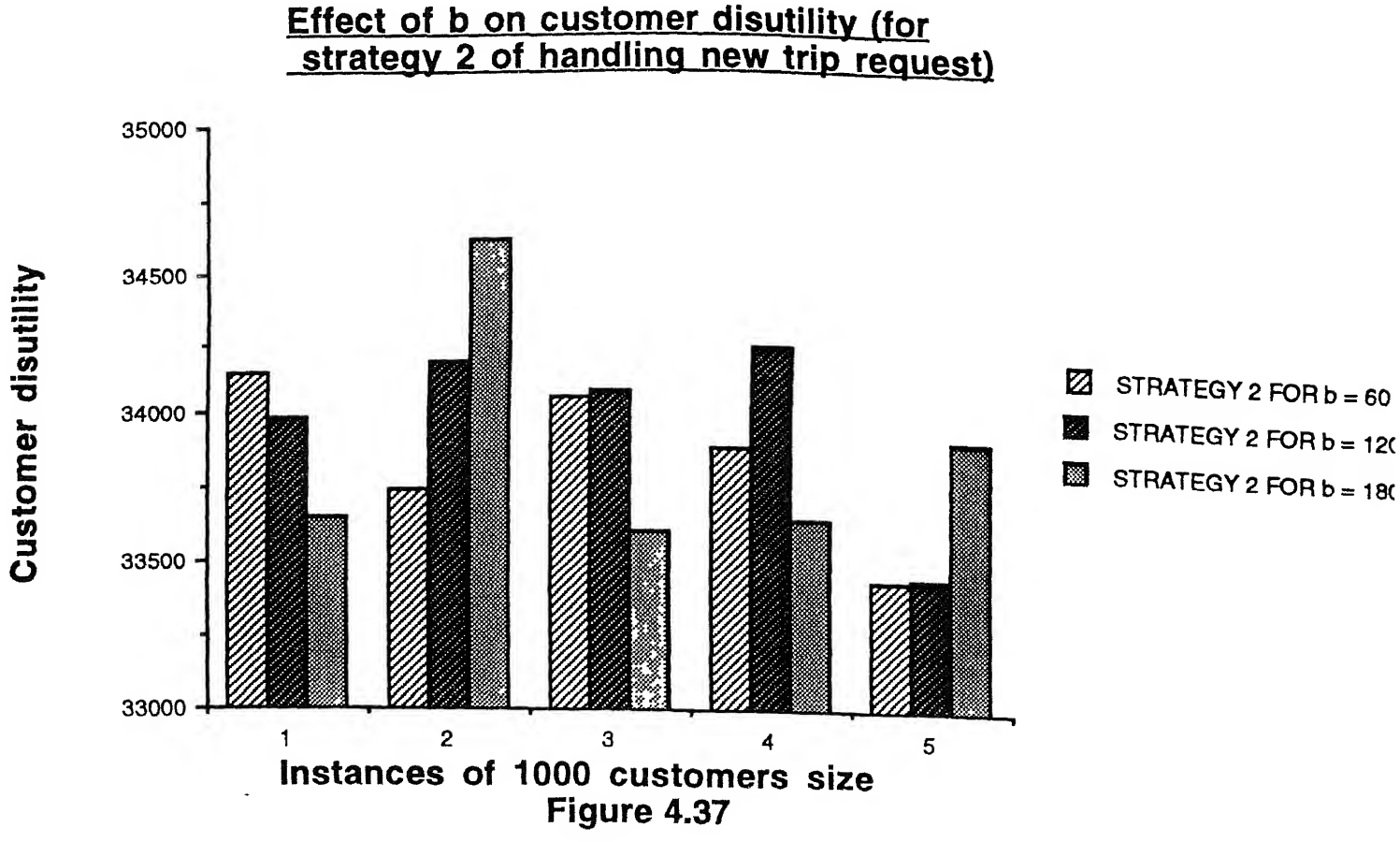

Effect of b on customer disutility (for strategy 2 of handling new trip request)
Customer disutility
35000
34500
34000
33500
33000
1
2
3
4
5
Instances of 1000 customers size
STRATEGY 2 FOR b = 60
STRATEGY 2 FOR b = 12(
STRATEGY 2 FOR b = 18(

Figure 4.37

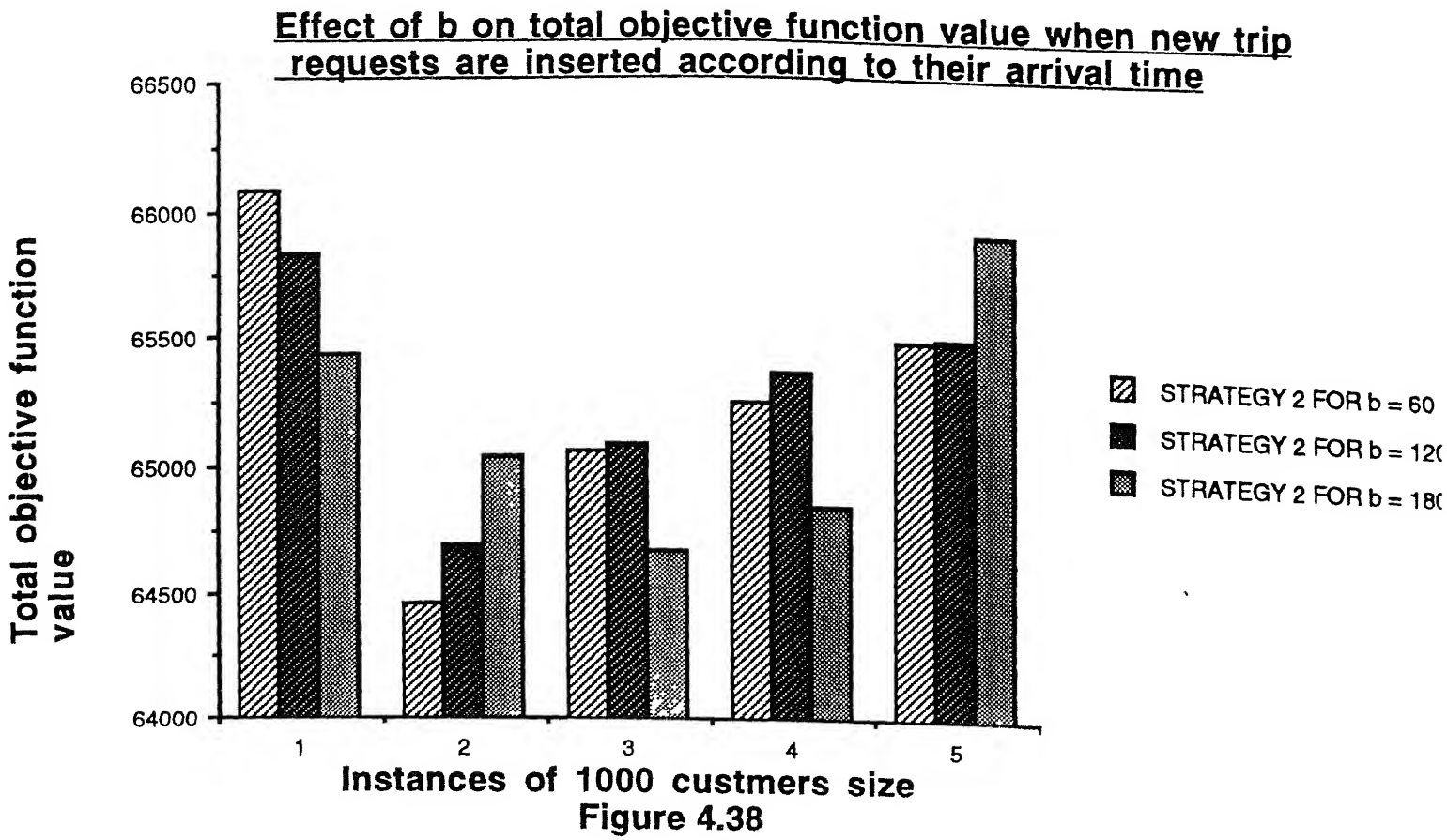

Effect of b on total objective function value when new trip requests are inserted according to their arrival time
Total objective function value
66500
66000
65500
65000
64500
64000
1
2
3
4
5
Instances of 1000 custmers size
STRATEGY 2 FOR b = 60
STRATEGY 2 FOR b = 12(
STRATEGY 2 FOR b = 18(

Figure 4.38

Comparison between customer disutility obtained by applying different strategies of handling new trip requests for b = 60

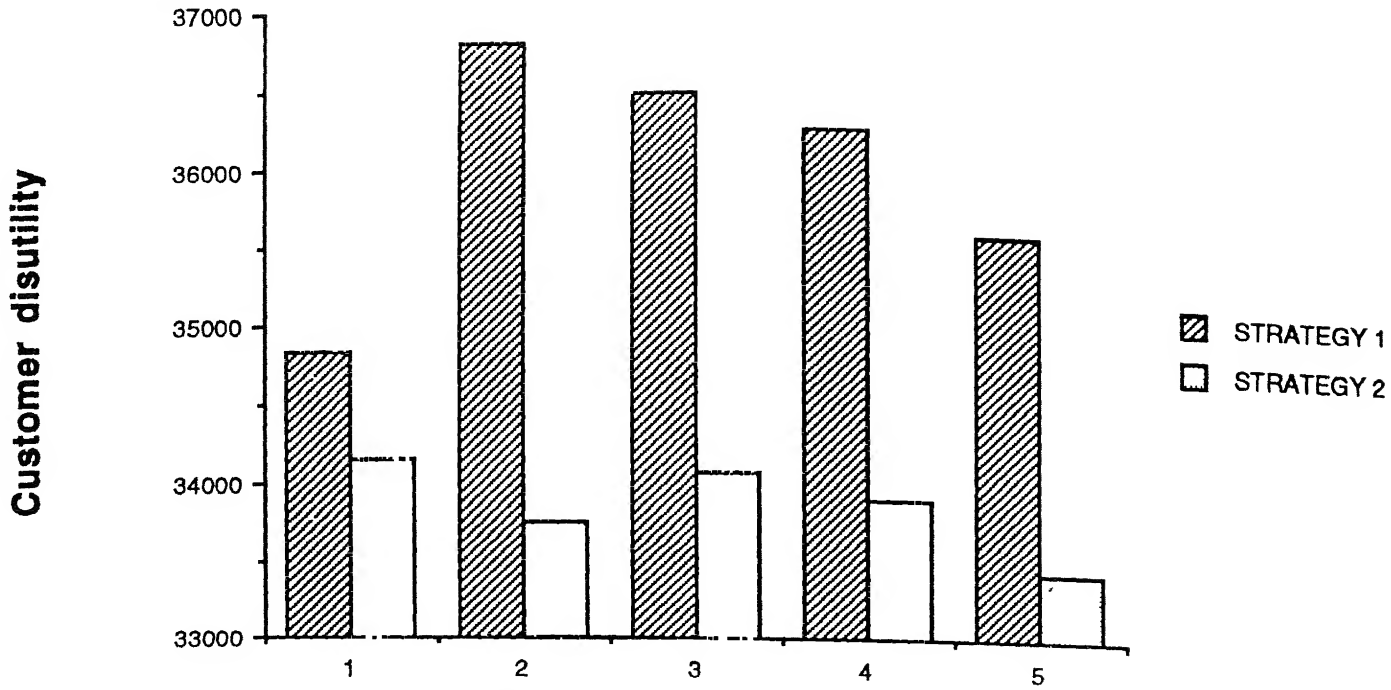


Figure 4.39

Comparison between total objective function value obtained for different strategies of handling new trip requests for b = 60

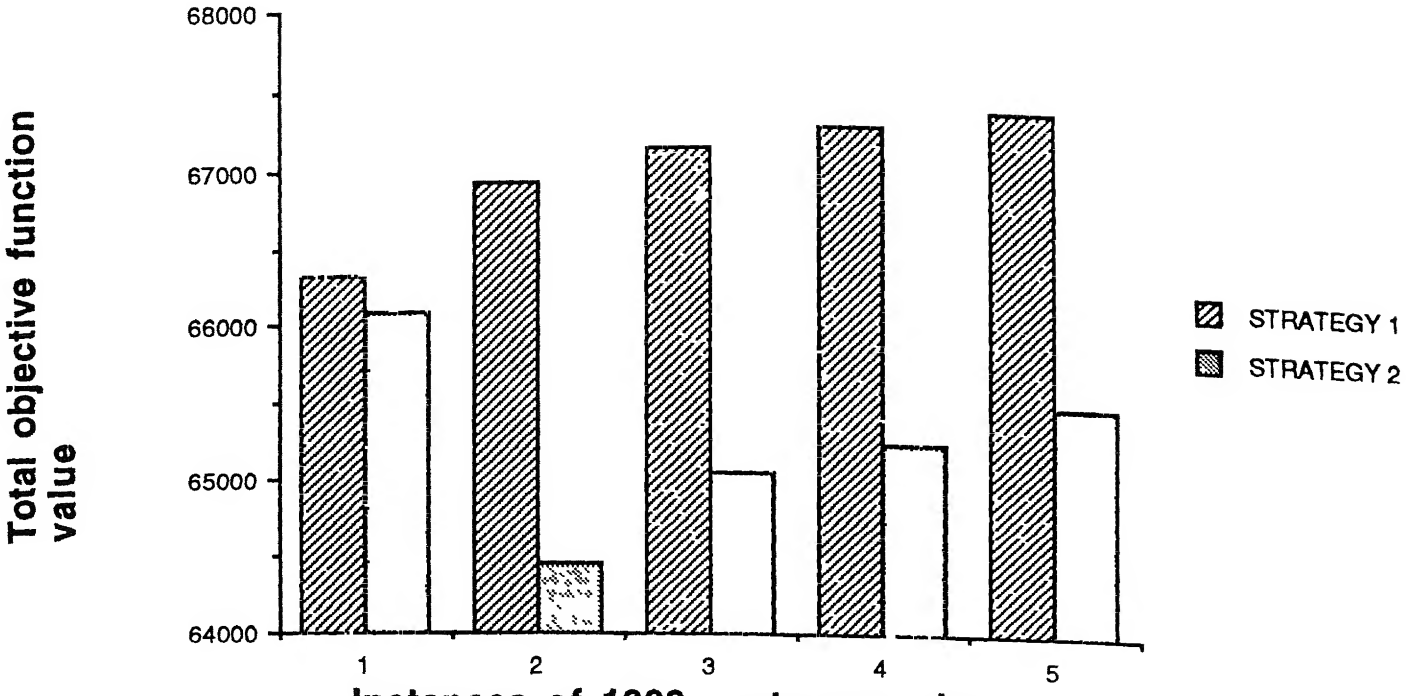


Figure 4.40

Comparison between customer disutility obtained by different strategies of handling immediate requests for b = 120

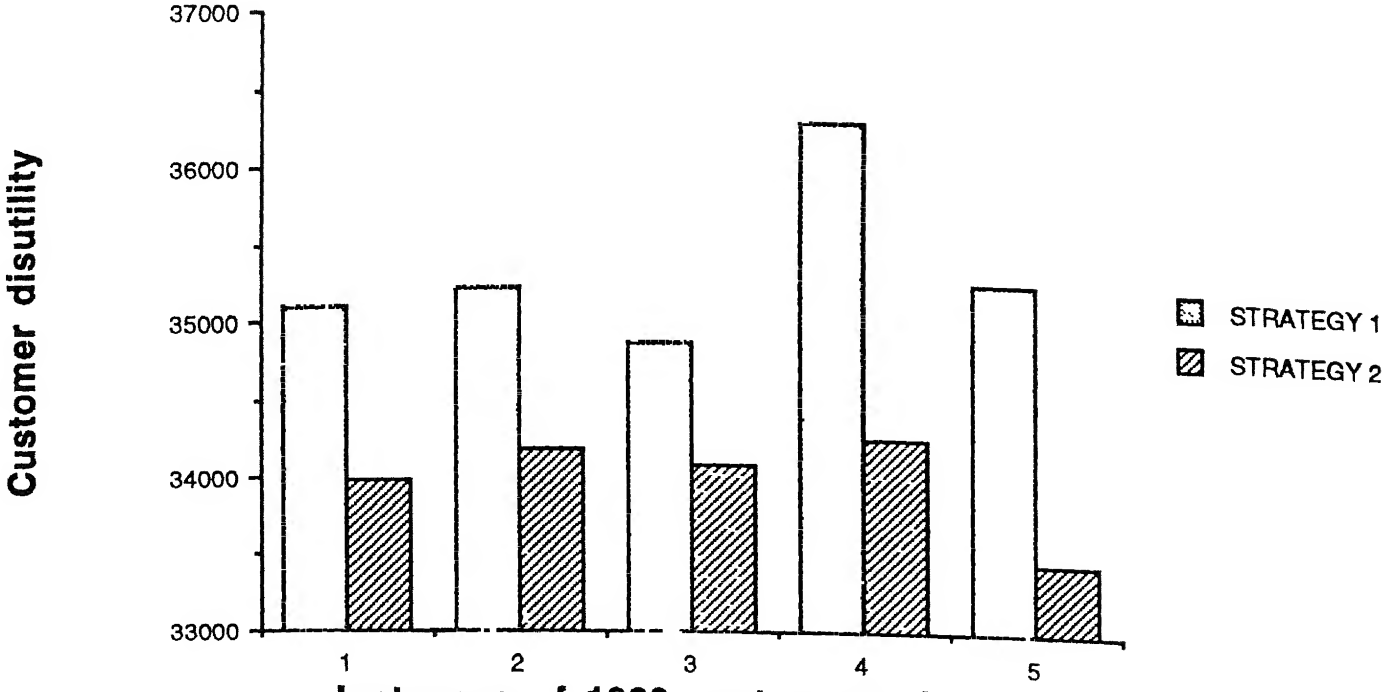


Figure 4.41

Comparison between customer disutility obtained by different strategies of handling immediate requests for b = 180

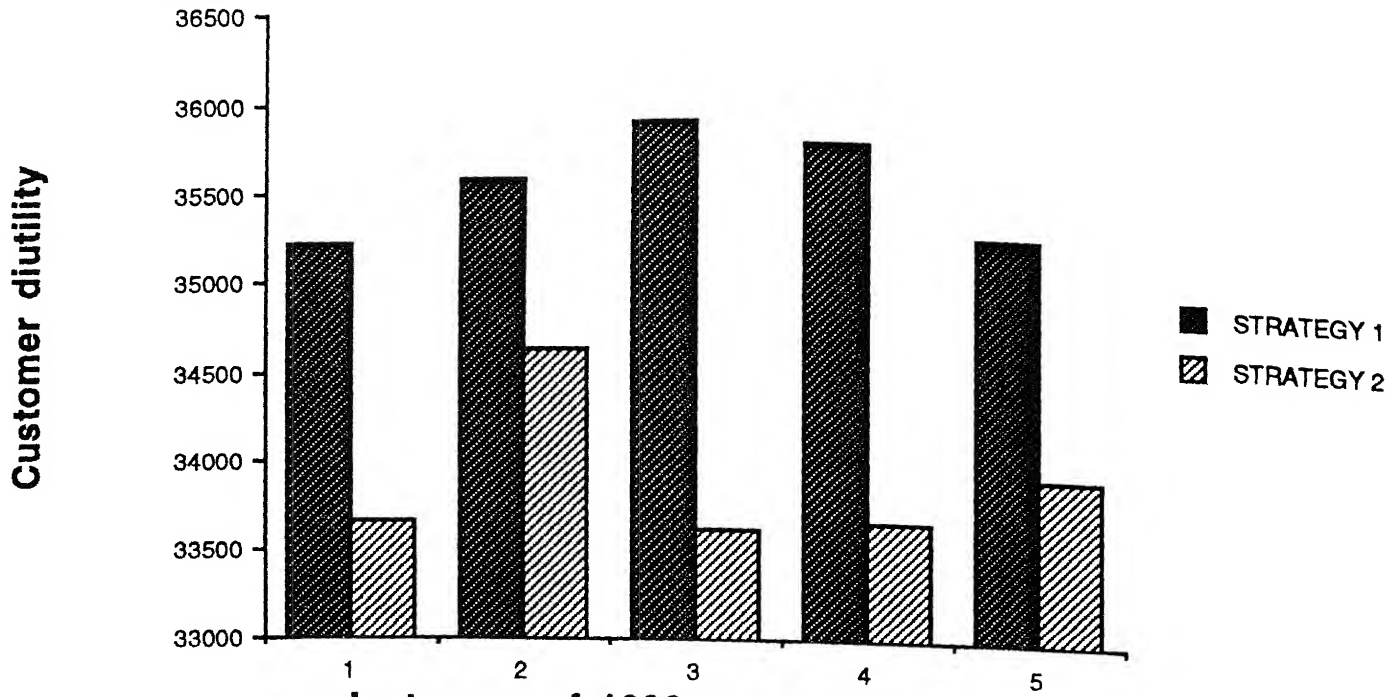


Figure 4.43

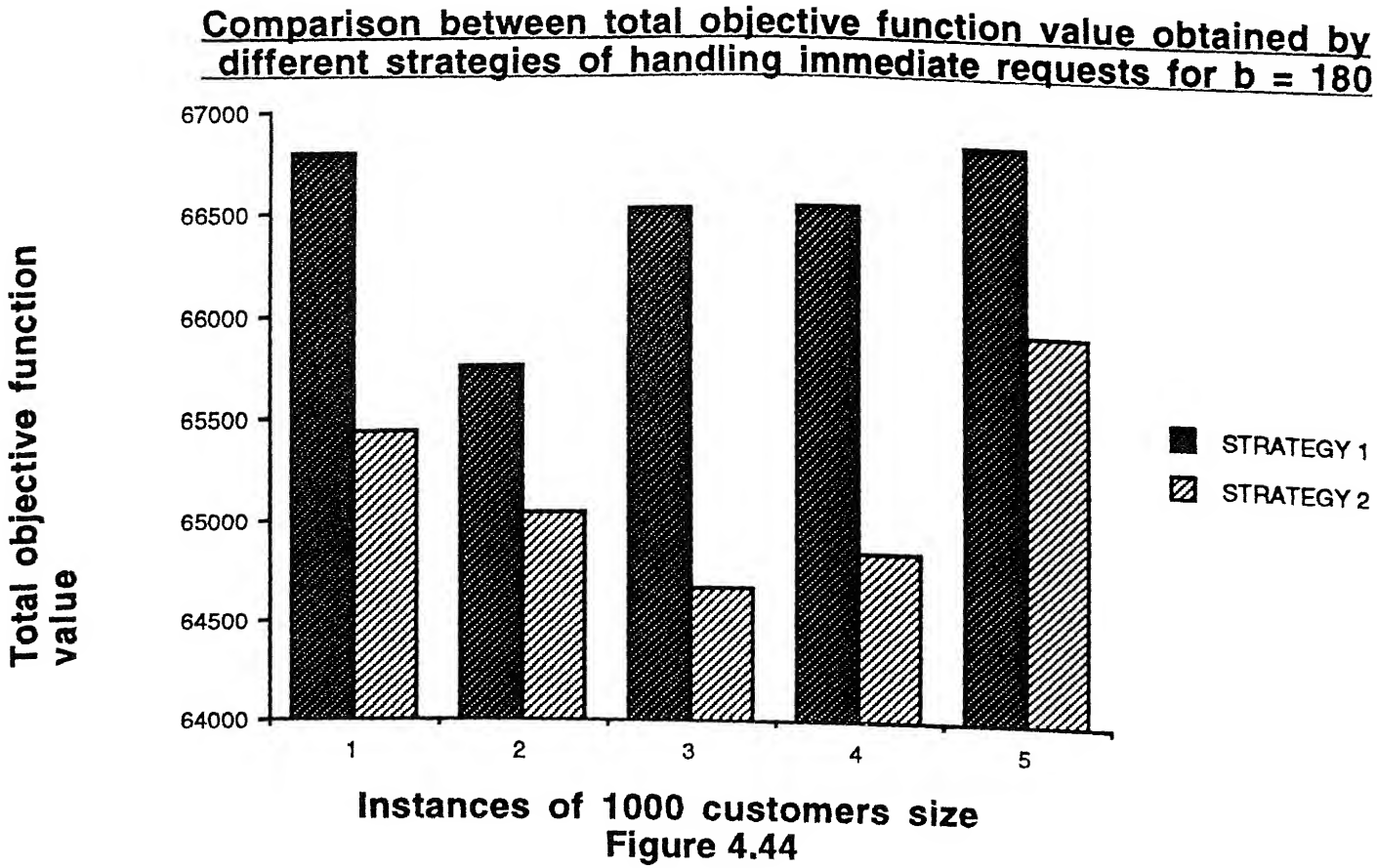

Comparison between total objective function value obtained by different strategies of handling immediate requests for b = 180
Total objective function value
67000
66500
66000
65500
65000
64500
64000
1
2
3
4
5
STRATEGY 1
STRATEGY 2
Instances of 1000 customers size

Figure 4.44

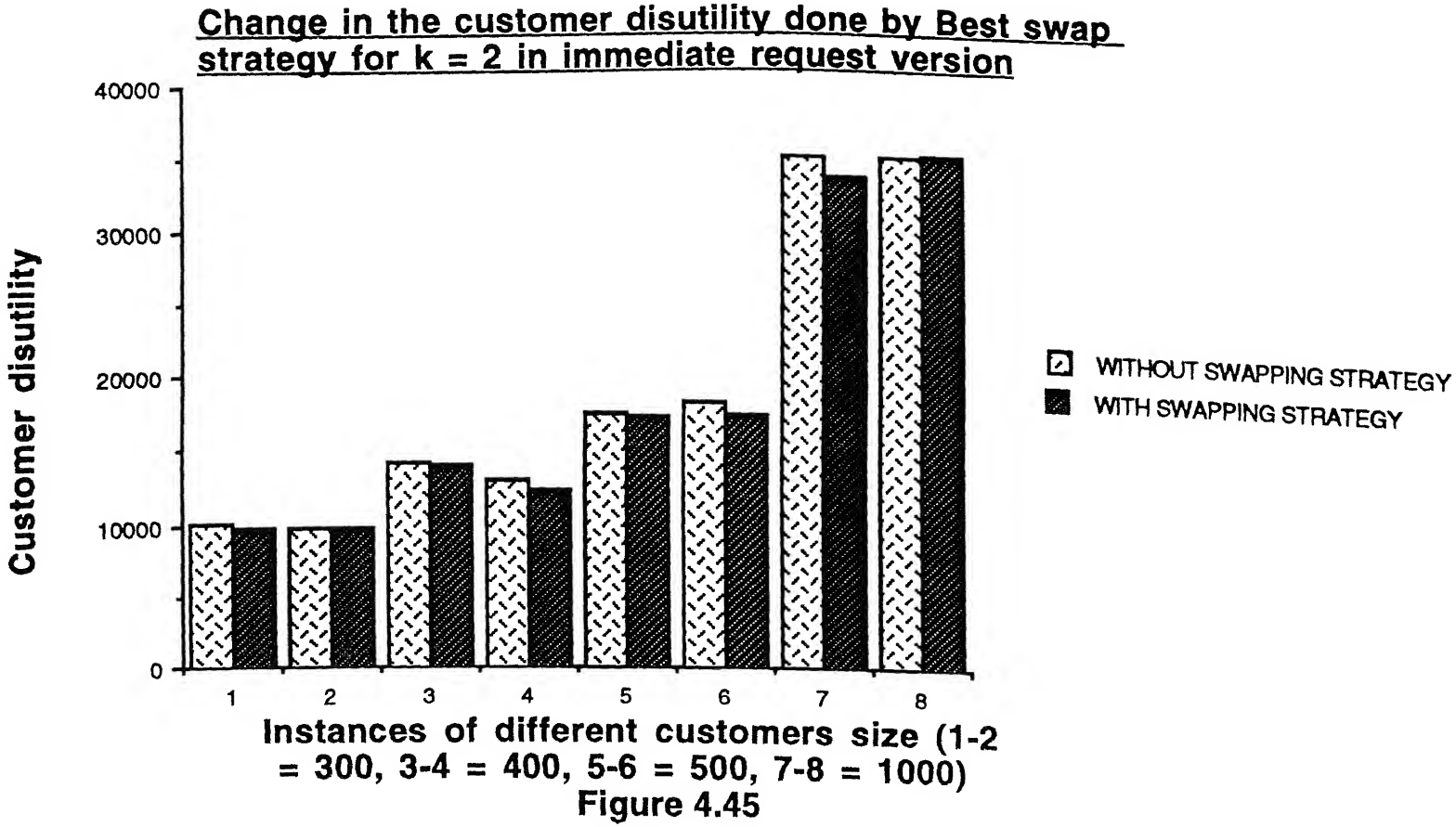

Change in the customer disutility done by Best swap strategy for k = 2 in immediate request version
Customer disutility
40000
30000
20000
10000
0
1
2
3
4
5
6
7
8
Instances of different customers size (1-2 = 300, 3-4 = 400, 5-6 = 500, 7-8 = 1000)
WITHOUT SWAPPING STRATEGY
WITH SWAPPING STRATEGY

Figure 4.45

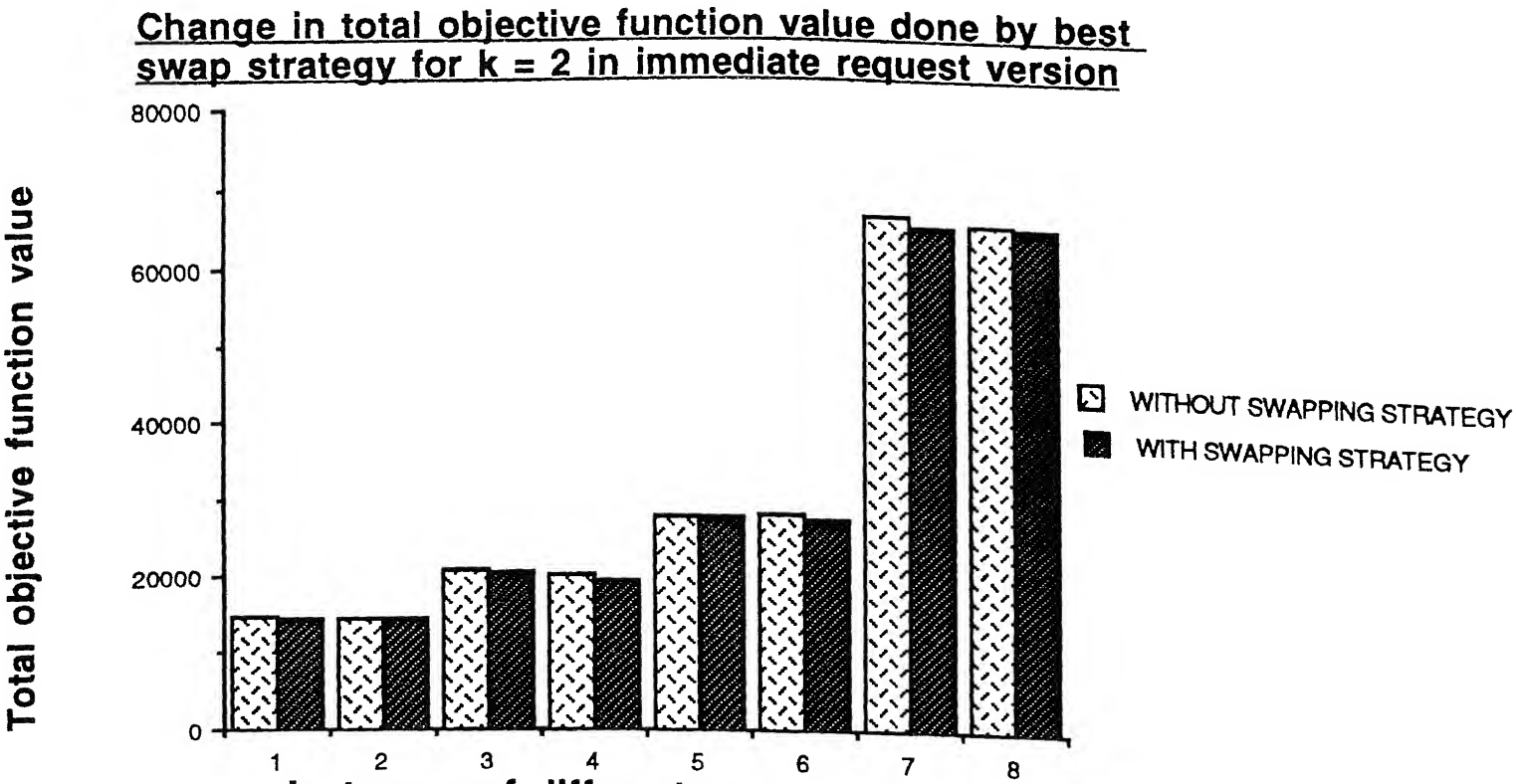

Change in total objective function value done by best swap strategy for k = 2 in immediate request version
Total objective function value
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
WITHOUT SWAPPING STRATEGY
WITH SWAPPING STRATEGY
Instances of different customers size (1-2 = 300, 3-4 = 400, 5-6 = 500, 7-8 = 1000)

Figure 4.46

strategy two of handling new trip requests for **b** = 60 to handle the immediate request).

The first approach is applicable if we know all the canceled request before the starting time of ADART service. We design it only to judge the performance of enumeration strategy to handle cancellations The second approach does not have this restriction. This is used to handle cancellation requests in simulation model. For this, we do enumeration at the arrival of cancellation requests. Cancellation requests should come **d** time before cancellation request's EPT, where **d** is a parameter which we can change.

We assume that if algorithms are not able to generate route without cancellation requests, then we will keep the route with cancellation request. In this case the taxi has to go to the cancellation request's origin and destination as they are not canceled.

Computational results

We take results for different value of **c**. Where **c** is the percentage of total advance trip requests canceled by customers. We can change the value of **c**. When we do cancellations in advance trip request version, we observe that algorithm is very successful in performing the cancellations, as shown in Figures 4.47 and 4 48. These Figures are showing results of two instances. But when we do the cancellations in immediate request version, we observe that the algorithm is able to do cancellations approximately 75% times as shown in Figure 4.49. We take the results for different values of **d**. We find **d** has insignificant effect on the cancellations done successfully by the algorithm. Our explanation for such behavior is that when we are doing cancellations for high value of **d** (i.e., d = 180), the number of nodes near cancellations requests in the route are less so we are not able to enumerate route without cancellation requests. For small value of **d**, the number of nodes near

Cancellations done successfully for different value of c in advance trip requests version using enumeration strategy

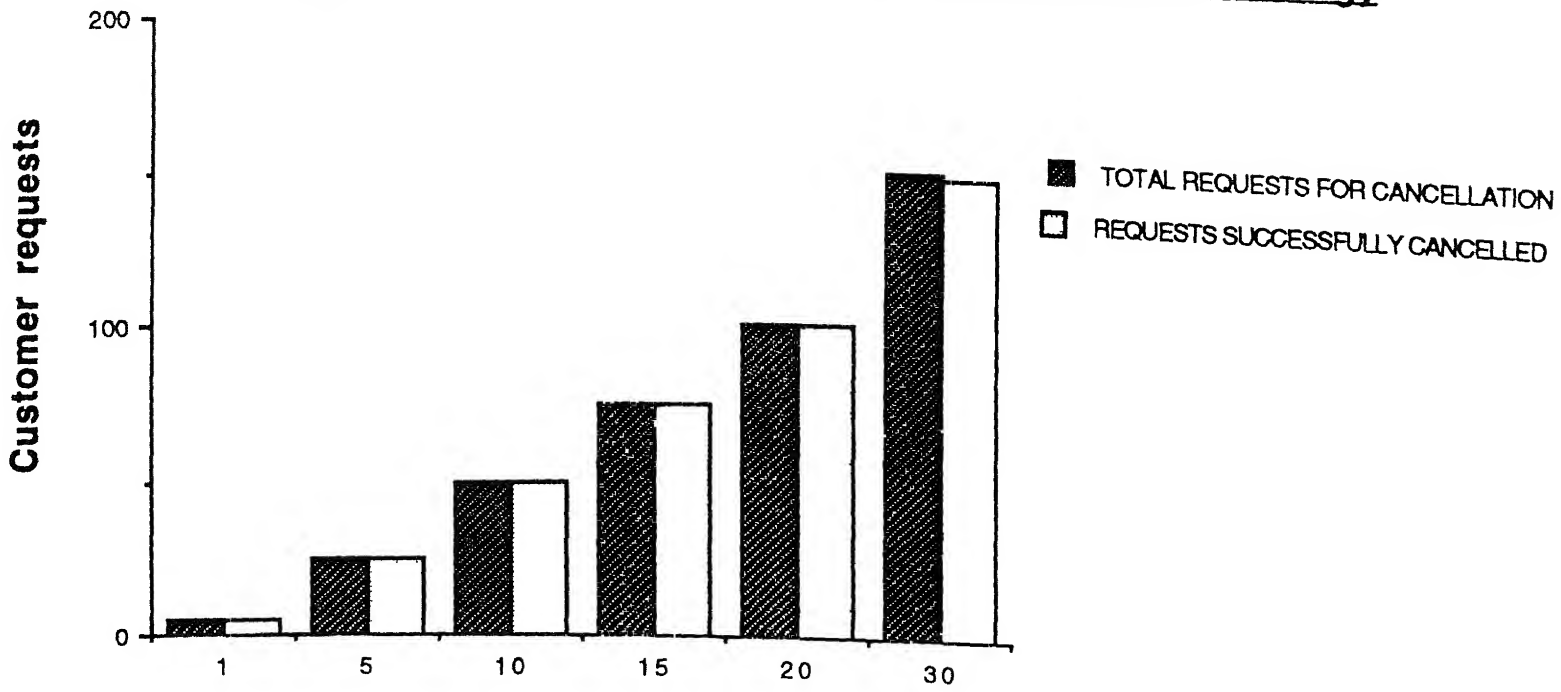


Figure 4.47

cancellations done successfully for different value of c in advance trip request version using enumeration strategy

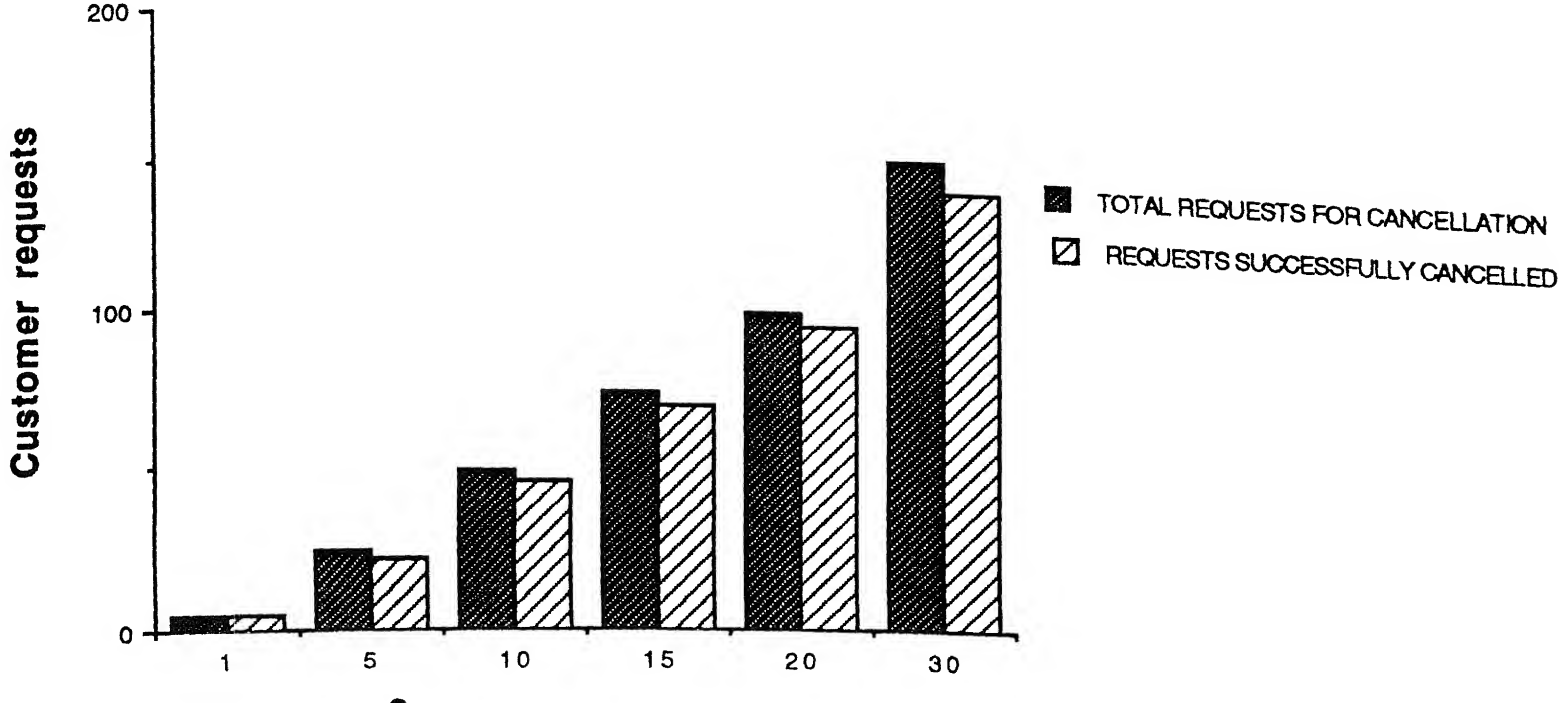


Figure 4.48

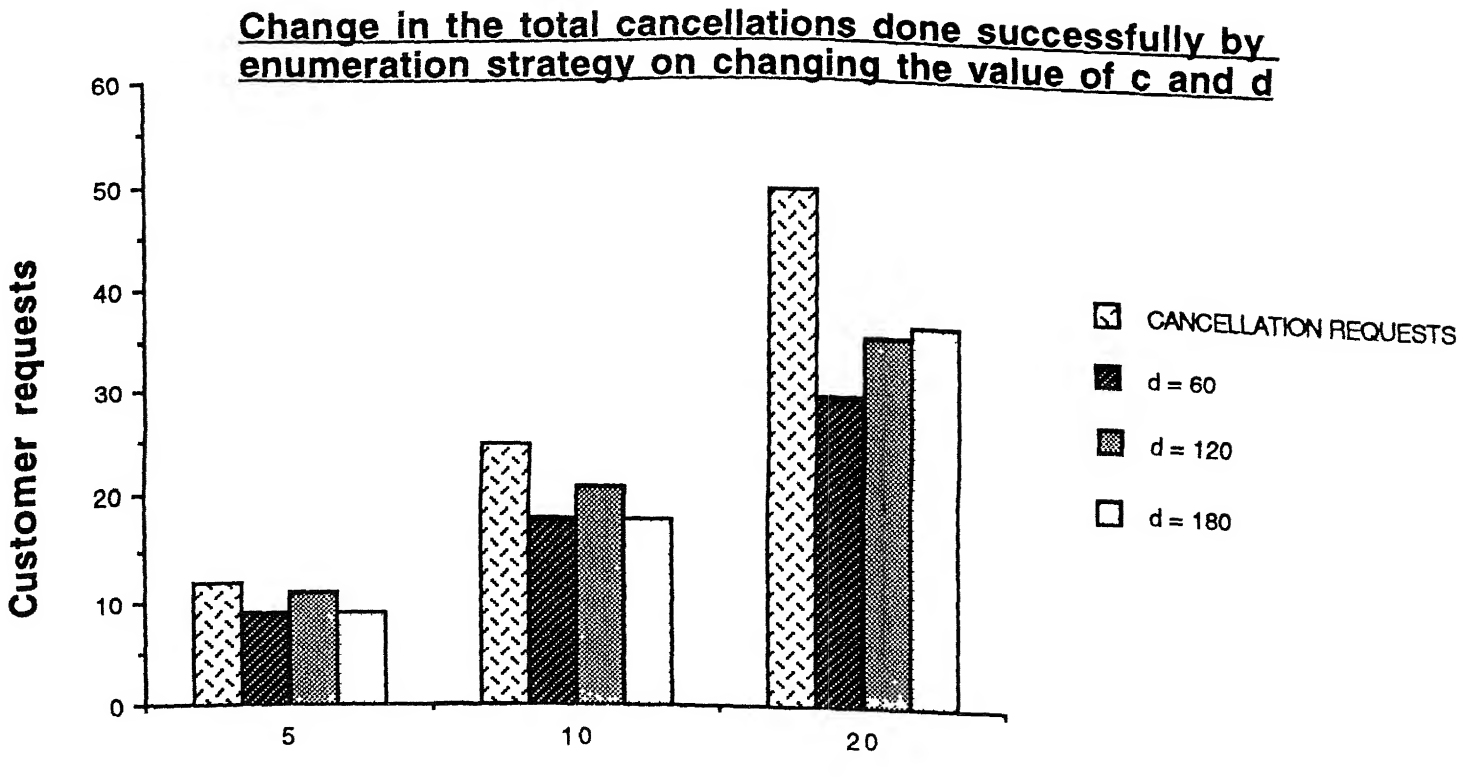

Change in the total cancellations done successfully by enumeration strategy on changing the value of c and d
Customer requests
60
50
40
30
20
10
0
5
10
20
c
CANCELLATION REQUESTS
d = 60
d = 120
d = 180

Figure 4.49

cancellations requests are more, but at the same time the number of nodes to enumerate decrease (due to late enumeration some nodes served by the vehicles does not take part in enumeration).

4.5 TAXI SIMULATION

After making all algorithms the idea came to our mind is that how much the ADART service beneficial than ordinary taxi service? This benefit is very important to support the idea of ADART. So to compare the performance of the ADART service and private taxi, we made the algorithm for private taxi (see Figure A.19). Kapil [5] did the private taxi simulation for advance request, we tried it for the immediate requests. For ADART service, some request are advance requests, and others are immediate requests. But for the taxi simulation, all the requests are immediate requests.

In private taxi, the customer do not share taxi with others. So the customer can be added only at the end of the route but in ADART system customers can share taxi with others so it promises better productivity of customers and taxi.

Computational results

We compared private taxi with our best model of ADART service. In ADART service, first we run ADARTW algorithm to get the initial solution, then we apply two run strategy. After that we improve the solution by strategy four of moving customers among routes. After that we improve the solution locally by enumeration strategy for $x = 2$. For handling new trip requests, we apply the second strategy of handling new trip requests for $b = 60$ minutes, and we keep on improving the solution by running the best swap strategy for $k = 2$ after insertion of every customer.

We take results for instances of 1000 customers having 0%, 25%, 50%, 75%, and 100% of many to few (MTF) type customer requests. For ADART service, we assume that 50% customer requests are advance trip requests, and 50% are the new trip requests. For private taxi, all customer requests are immediate requests. Figure 4.50 shows that as the percentage of MTF type customer requests increase, the ADART service performs well because vehicles serve more customers at a time on increasing the MTF type customer requests. We observe that customer disutility for ADART service is less than private taxi. This is a clear indication of the effectiveness of ADART system.

Total vehicles required by the private taxi increase as the percentage of MTF type customer requests increase. At the same time, total vehicles required by the ADART service reduce as the percentage of MTF type customer requests increase as shown in Figure 4.51.

The active time of vehicles on the road for private taxi are far more than ADART service as shown in Figure 4.52. Our explanation for this is that, in ADART service, more than one customer are served by the vehicle at a time So the time required to give service to all customers for ADART service is less than private taxi service.

To see the effect of ratio of the advance trip requests and immediate requests on ADART performance, we take results by varying the percentage of advance requests on one instance of 1000 customers having 50% of MTF type customers. Figure 4.53 shows that as the percentage of advance trip requests increase, the ADART service performs well.

Total vehicles required by the private taxi are approximately double than the ADART service as shown in Figure 4.54. The total number of vehicles required by the ADART service reduce as we increase the percentage of the advance trip requests.

Graph showing customer disutility found in case of ADART (having 50% advance trip requests), and private taxi

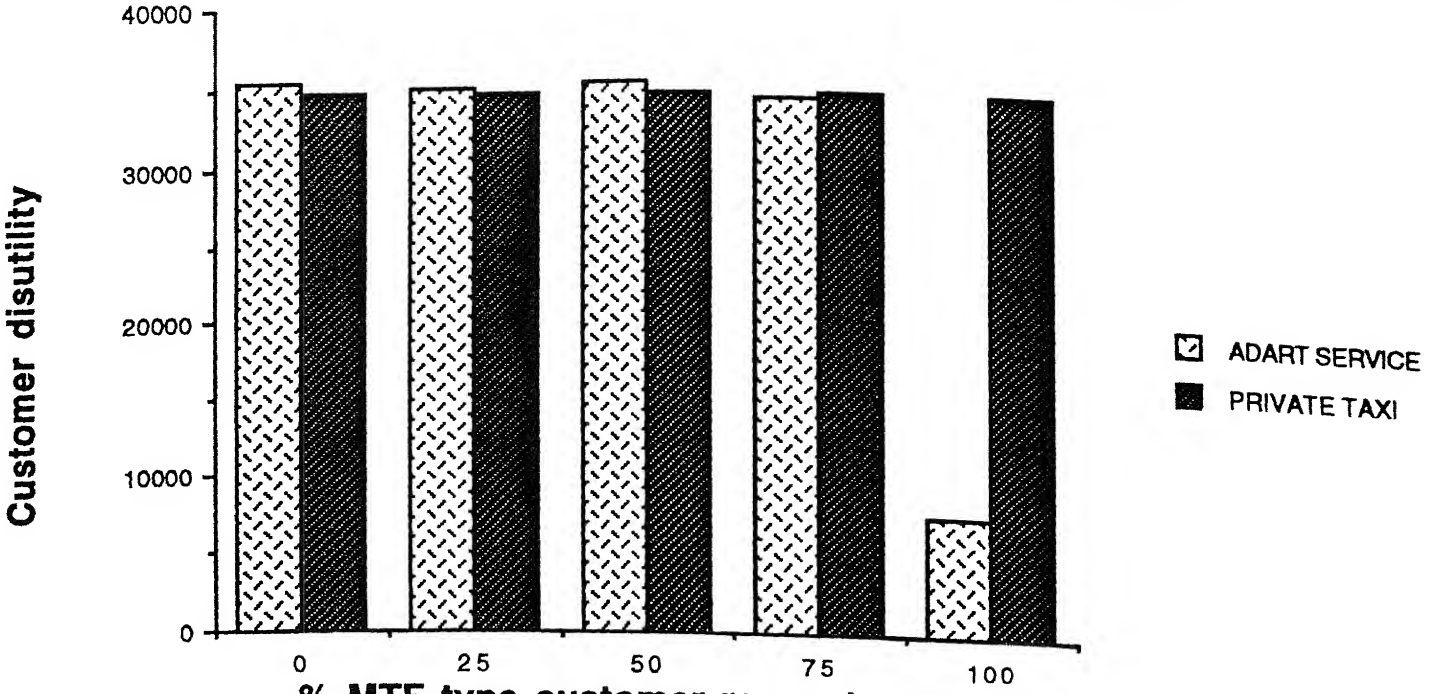


Figure 4.50

Graph showig comparison between vehicles required for ADART service (having 50% advance trip requests) , and for private taxi

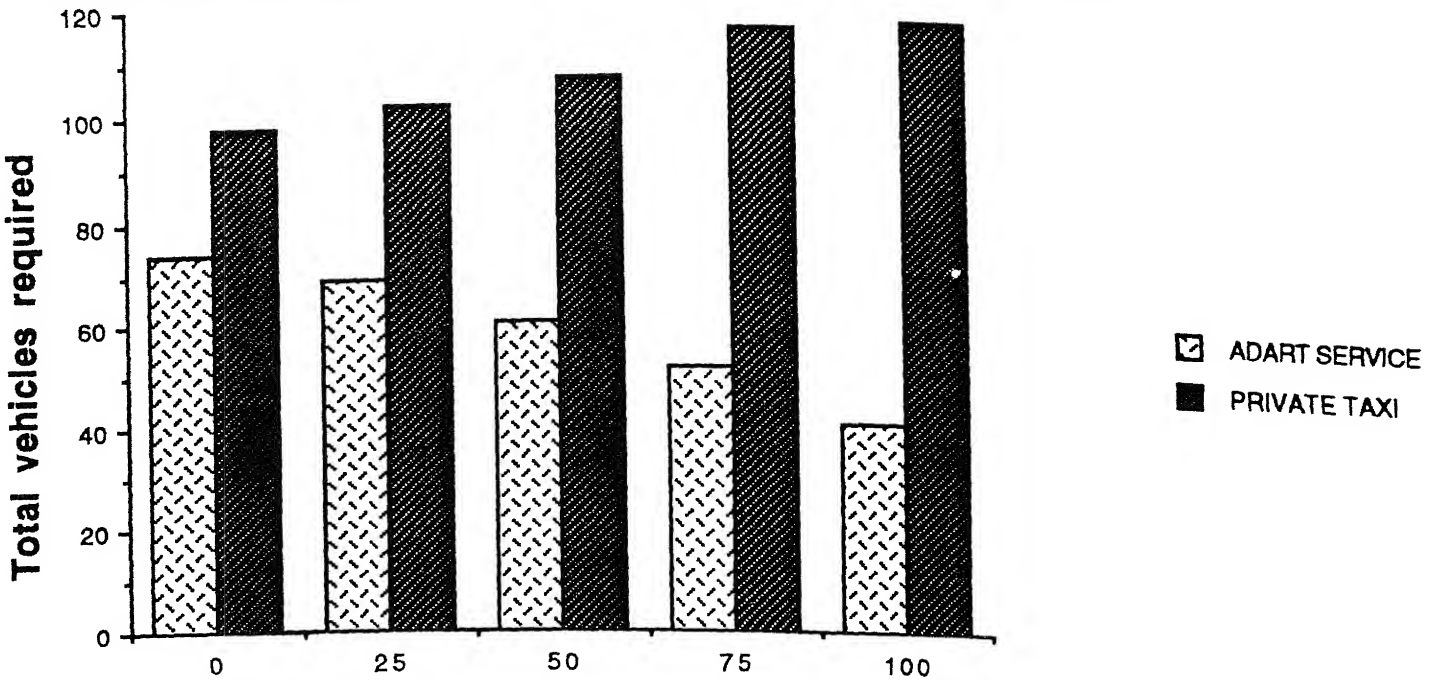


Figure 4.51

Comparion between the total service time required for ADART service(having 50% advance trip requets), and for private taxi

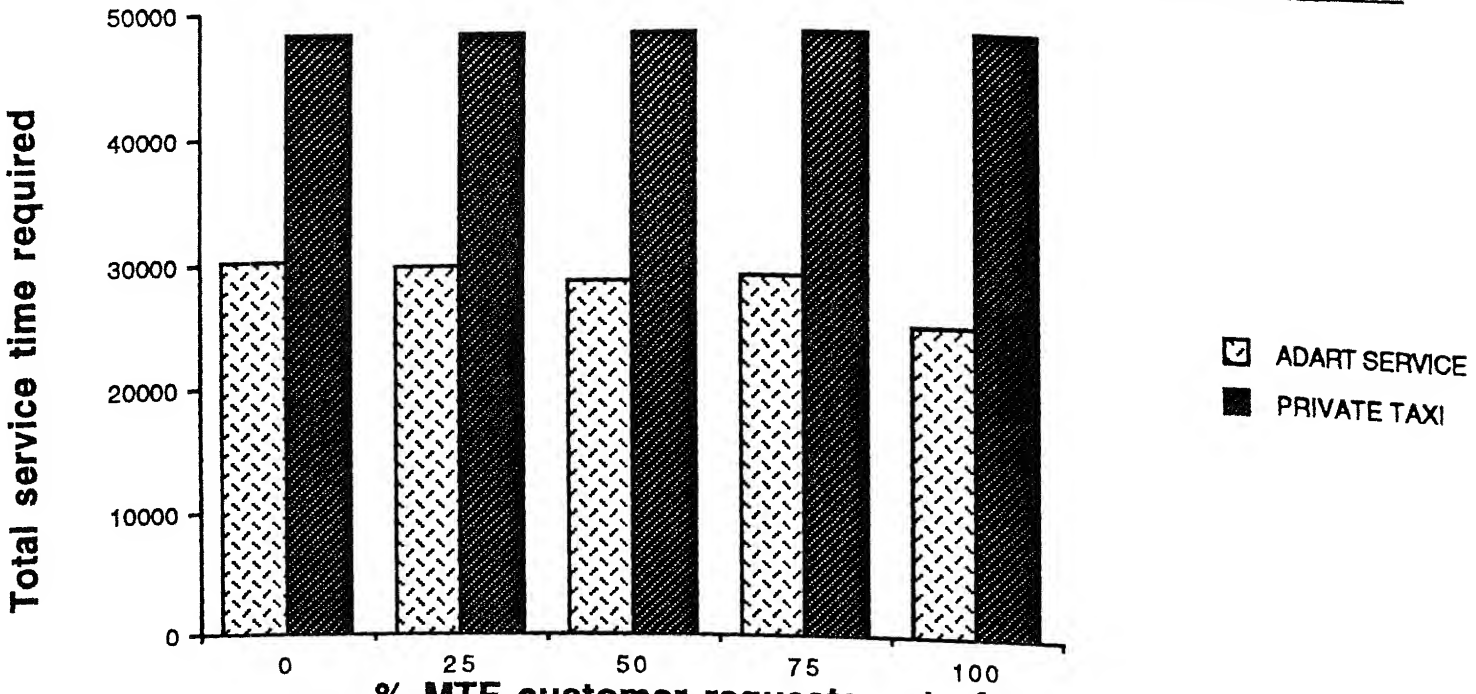


Figure 4.52

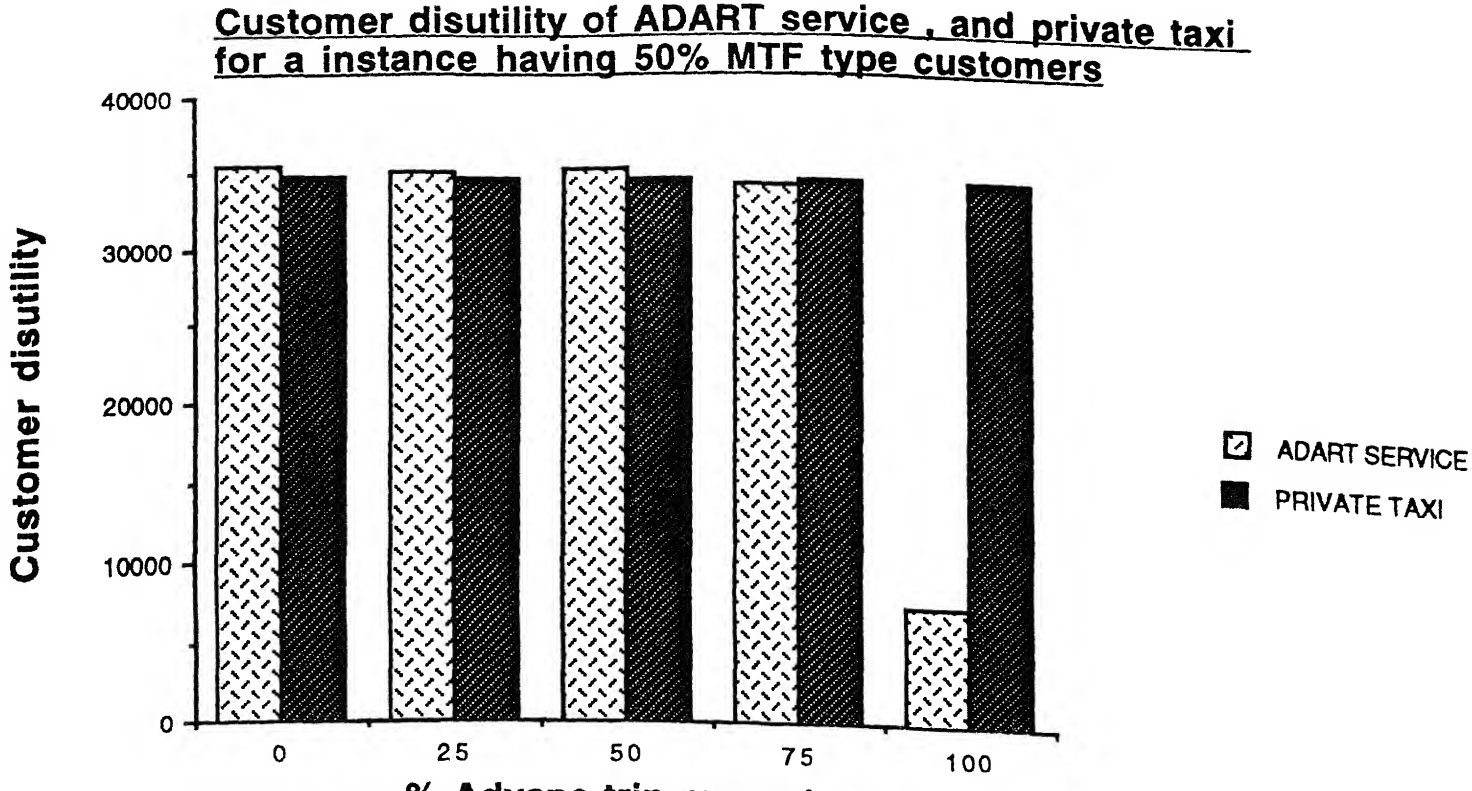

Customer disutility of ADART service , and private taxi for a instance having 50% MTF type customers
Customer disutility
40000
30000
20000
10000
0
0
25
50
75
100
% Advane trip requests out of 1000 customer requests
ADART SERVICE
PRIVATE TAXI

Figure 4.53

Graph showing the total vehicles required for ADART service, and for private taxi for a instance having 50% MTF type customer requests

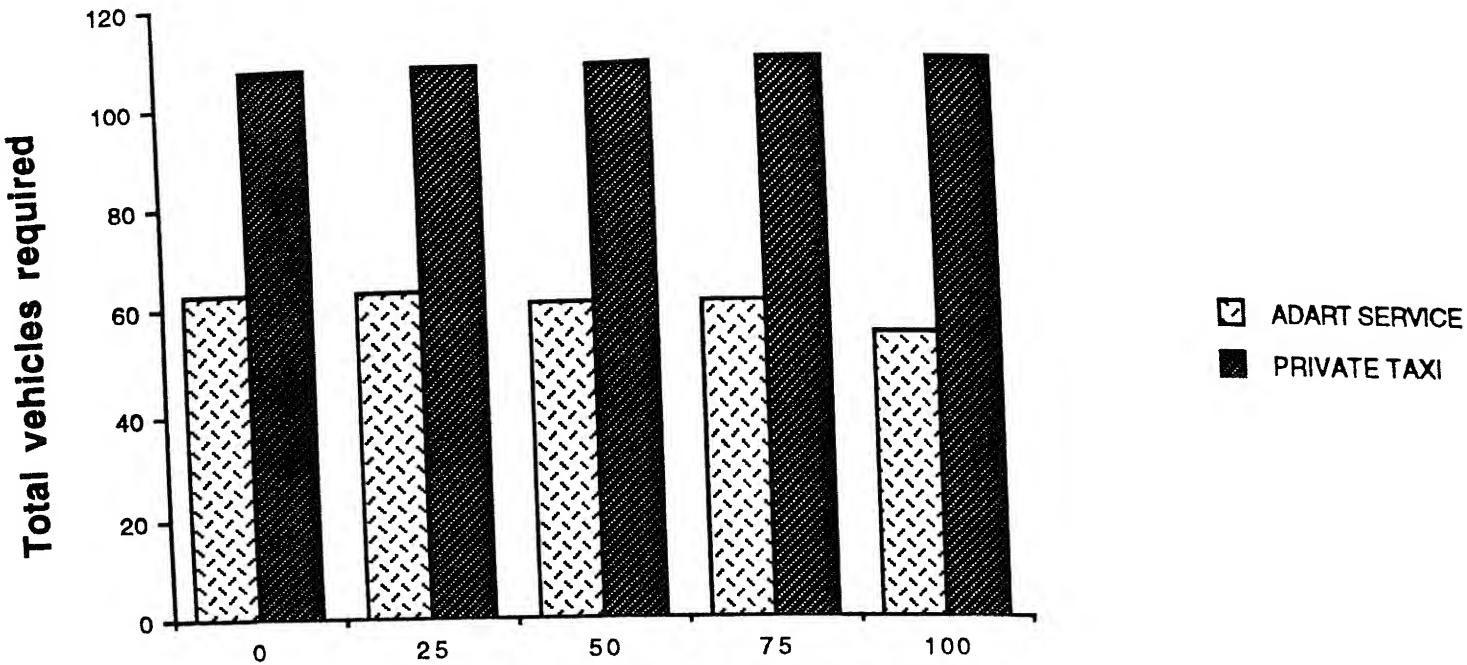


Figure 4.54

The active time on the road does not change much if we increase the percentage of the advance trip requests. But for ADART service the service time required by vehicles are far less than private taxi service.

4.6 AVENUES FOR FUTURE WORK

We try to improve the ADARTW solution by designing different local and global improvement schemes Various other strategies can also improve the solution further. The avenue for future work are as follows :

1. We use enumeration scheme which is using ADARTW algorithm for finding the feasible insertions of a customer in different routes. ADARTW algorithm can be improved further to get more good feasible ways of inserting a customer in a route. If one will improve them, the performance of enumeration strategy will be enhanced.

2. In our improvement strategies, we maintain feasibility of the whole route, but it is possible to improve the route by allowing infeasibility in the route for some time. One can think of improving the routes by allowing insertions and movement of customers which create infeasibility in the route, and make the route feasible by other customer insertion or movement of customers to other vehicles.

APPENDIX A

Table A.1 : Notation list (not defined in the text) for Figure A.1-A.19

start node : first node of the route

last node : last node of the route.

ST_j : Scheduled visit time of stop j in the schedule block before customer i is inserted

ET_j : Earliest permissible time to visit node j.

LT_j : Latest permissible time to visit node j.

VLT : Vehicles latest availability time.

v(-v) : vehicle origin(destination).

D(i,j) : Time required to travel from station i to station j.

$SLACK\_PREV_j$: slack available before the schedule block, having node j.

starting service time : The time at which the ADART service starts serving customers.

Ending service time : The time upto which the ADART service is available.

Present time : The current time of the day.

Interval : The time interval after which the algorithm for handling new trip requests is run by vehicle's computer.

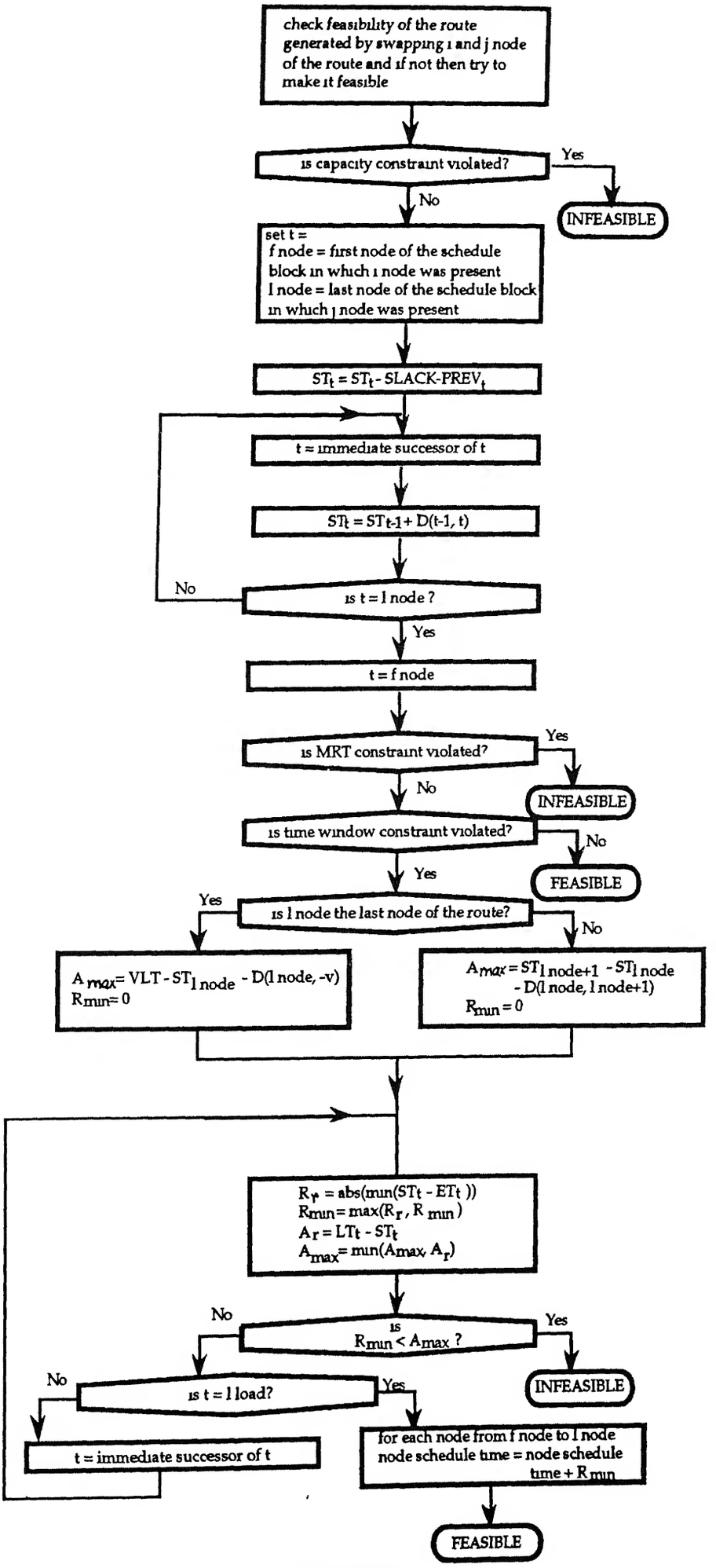

check feasibility of the route generated by swapping i and j node of the route and if not then try to make it feasible
is capacity constraint violated?
Yes
INFEASIBLE
No
set t =
f node = first node of the schedule block in which i node was present
l node = last node of the schedule block in which j node was present
$ST_t = ST_t$ - SLACK-PREV$_t$
t = immediate successor of t
$ST_t = ST_{t-1} + D(t-1, t)$
No
is t = l node ?
Yes
t = f node
is MRT constraint violated?
Yes
INFEASIBLE
No
is time window constraint violated?
No
FEASIBLE
Yes
Yes
is l node the last node of the route?
No
A_{max}= VLT - $ST_{l\ node}$ - D(l node, -v)
R_{min}= 0
$A_{max} = ST_{l\ node+1} - ST_{l\ node}$ - D(l node, l node+1)
$R_{min} = 0$
R_r = abs(min(ST_t - ET_t))
R_{min}= max(R_r, R_{min})
$A_r = LT_t - ST_t$
A_{max}= min(A_{max}, A_r)
No
is $R_{min} < A_{max}$?
Yes
No
is t = l load?
Yes
INFEASIBLE
t = immediate successor of t
for each node from f node to l node
node schedule time = node schedule time + R_{min}
FEASIBLE

Figure A.1

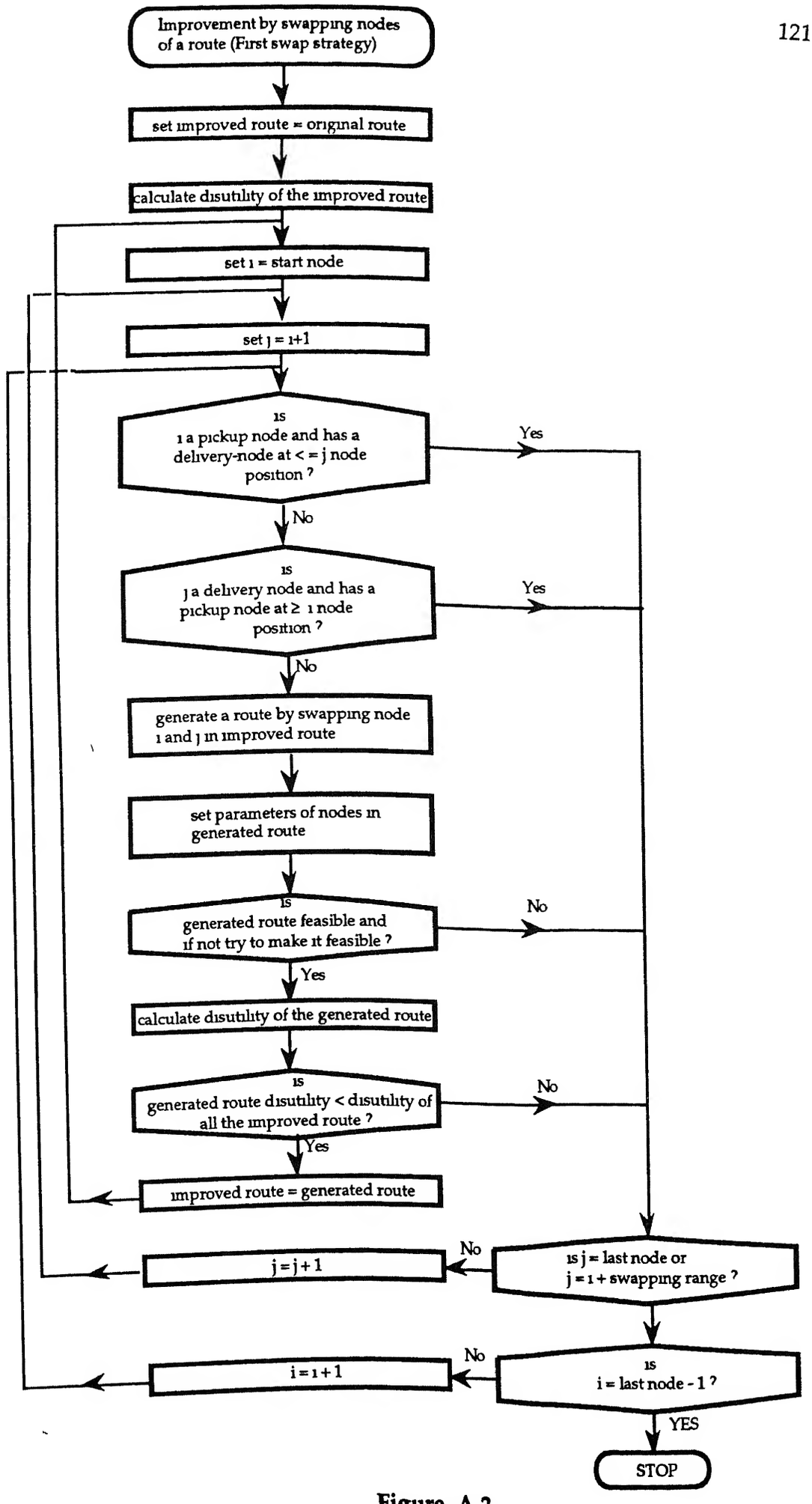

Improvement by swapping nodes of a route (First swap strategy)
set improved route = original route
calculate disutility of the improved route
set i = start node
set j = i+1
is i a pickup node and has a delivery-node at < = j node position ?
Yes
No
is j a delivery node and has a pickup node at ≥ i node position ?
Yes
No
generate a route by swapping node i and j in improved route
set parameters of nodes in generated route
is generated route feasible and if not try to make it feasible ?
No
Yes
calculate disutility of the generated route
is generated route disutility < disutility of all the improved route ?
No
Yes
improved route = generated route
is j = last node or j = i + swapping range ?
No
j = j + 1
is i = last node - 1 ?
No
i = i + 1
YES
STOP

Figure A.2

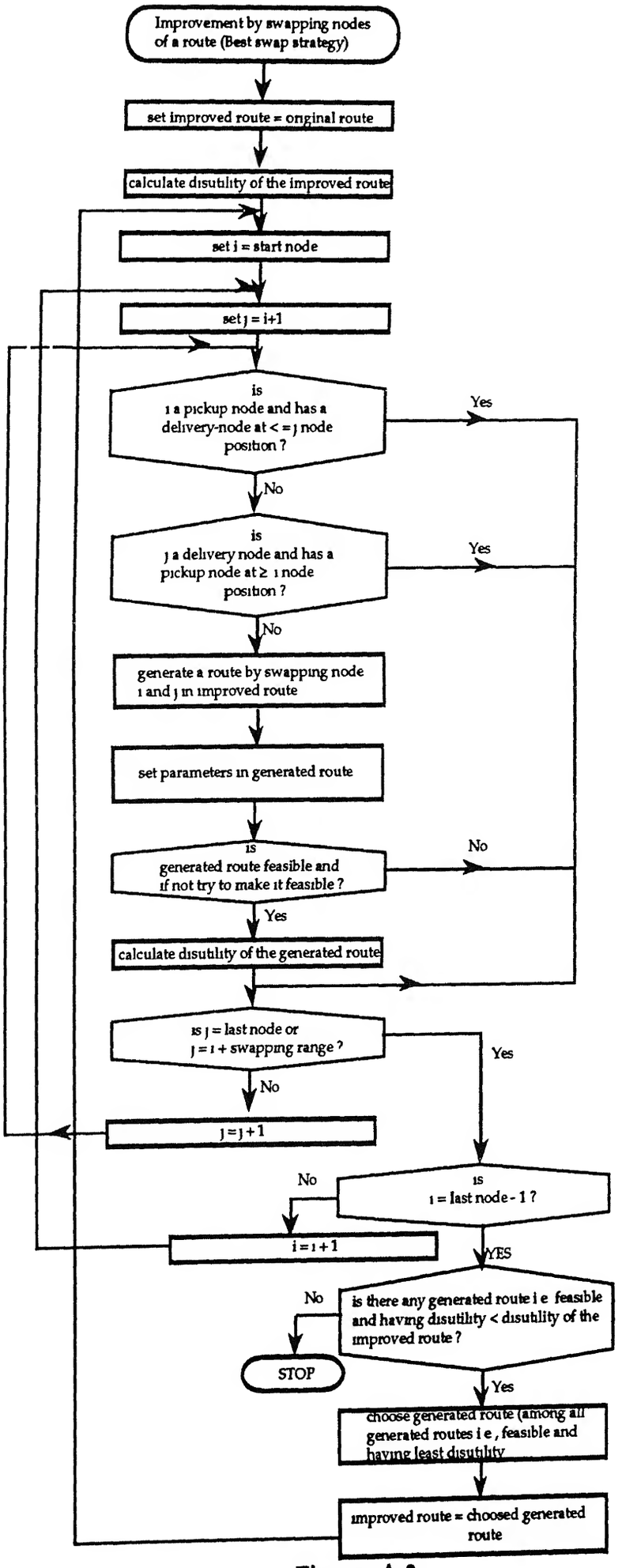

Improvement by swapping nodes of a route (Best swap strategy)
set improved route = original route
calculate disutility of the improved route
set i = start node
set j = i+1
is i a pickup node and has a delivery-node at < = j node position ?
Yes
No
is j a delivery node and has a pickup node at ≥ i node position ?
Yes
No
generate a route by swapping node i and j in improved route
set parameters in generated route
is generated route feasible and if not try to make it feasible ?
No
Yes
calculate disutility of the generated route
is j = last node or j = i + swapping range ?
Yes
No
j = j + 1
No
is i = last node - 1 ?
i = i + 1
YES
No
is there any generated route i e feasible and having disutility < disutility of the improved route ?
STOP
Yes
choose generated route (among all generated routes i e , feasible and having least disutility
improved route = choosed generated route

Figure A.3

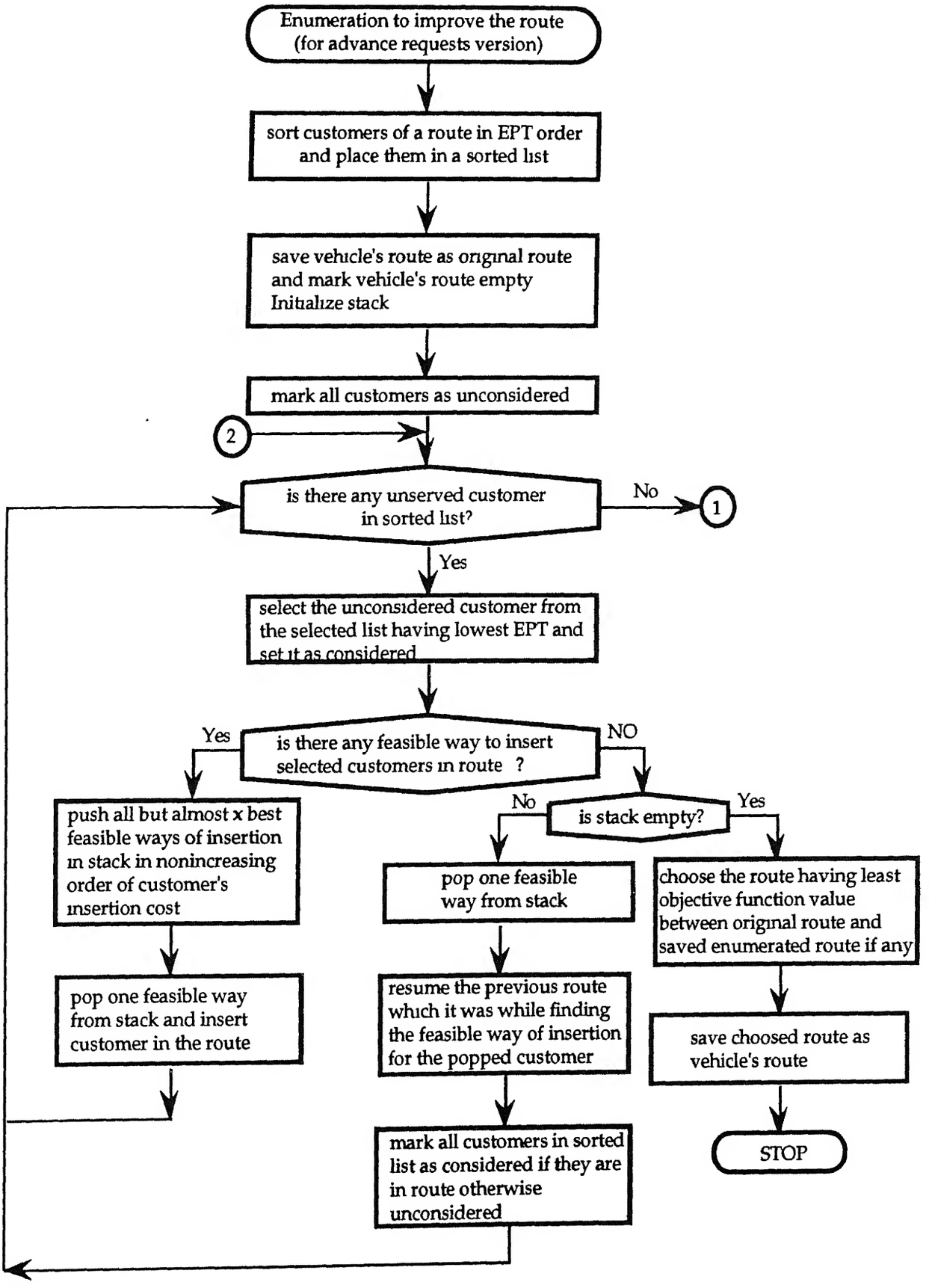

Enumeration to improve the route (for advance requests version)
sort customers of a route in EPT order and place them in a sorted list
save vehicle's route as original route and mark vehicle's route empty Initialize stack
mark all customers as unconsidered
2
is there any unserved customer in sorted list?
No
1
Yes
select the unconsidered customer from the selected list having lowest EPT and set it as considered
Yes
is there any feasible way to insert selected customers in route ?
NO
push all but almost x best feasible ways of insertion in stack in nonincreasing order of customer's insertion cost
pop one feasible way from stack and insert customer in the route
No
is stack empty?
Yes
pop one feasible way from stack
resume the previous route which it was while finding the feasible way of insertion for the popped customer
mark all customers in sorted list as considered if they are in route otherwise unconsidered
choose the route having least objective function value between original route and saved enumerated route if any
save choosed route as vehicle's route
STOP

Figure A.4

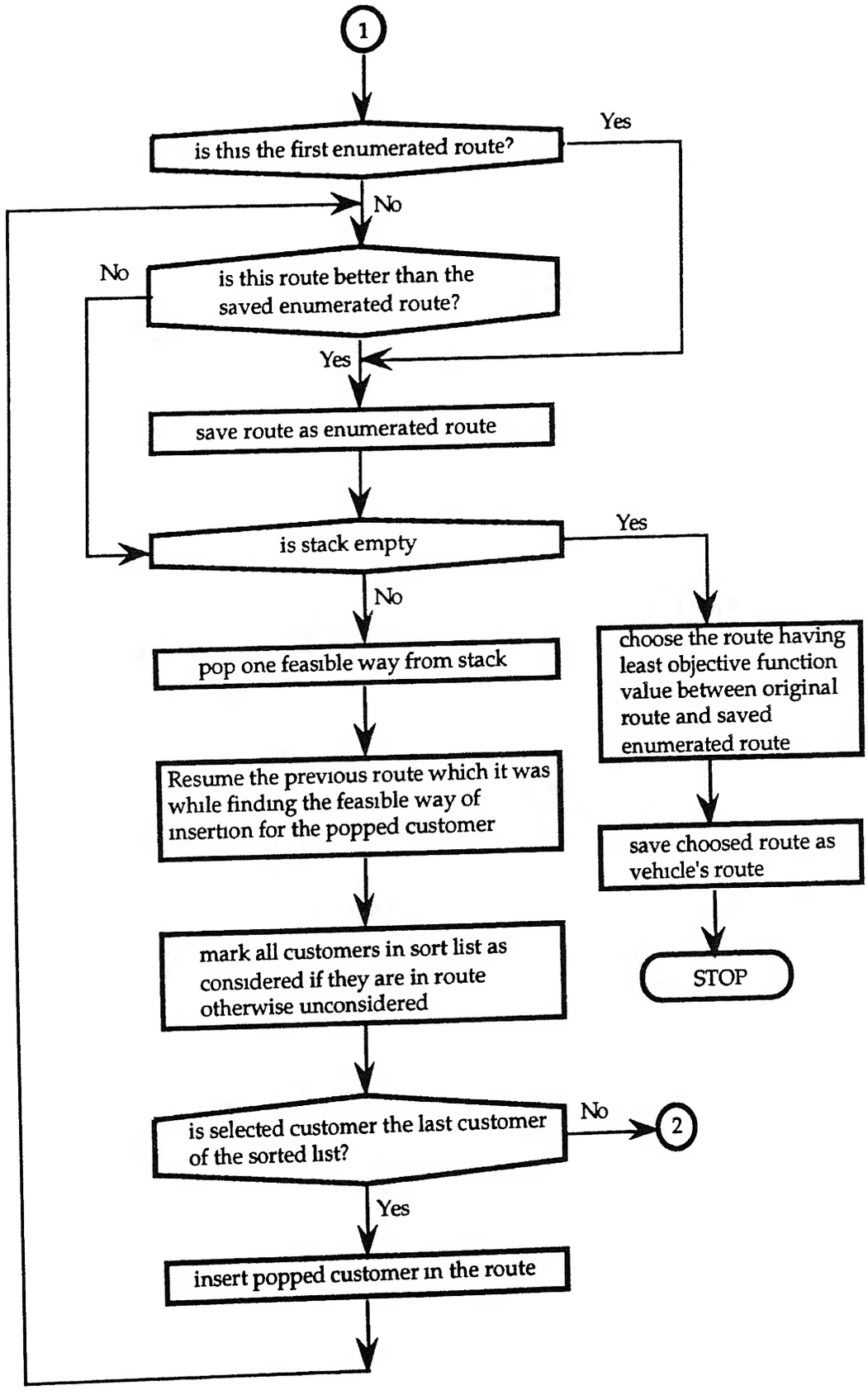

1
is this the first enumerated route?
Yes
No
is this route better than the saved enumerated route?
No
Yes
save route as enumerated route
is stack empty
Yes
No
pop one feasible way from stack
Resume the previous route which it was while finding the feasible way of insertion for the popped customer
mark all customers in sort list as considered if they are in route otherwise unconsidered
is selected customer the last customer of the sorted list?
No
2
Yes
insert popped customer in the route
choose the route having least objective function value between original route and saved enumerated route
save choosed route as vehicle's route
STOP

Figure A.5

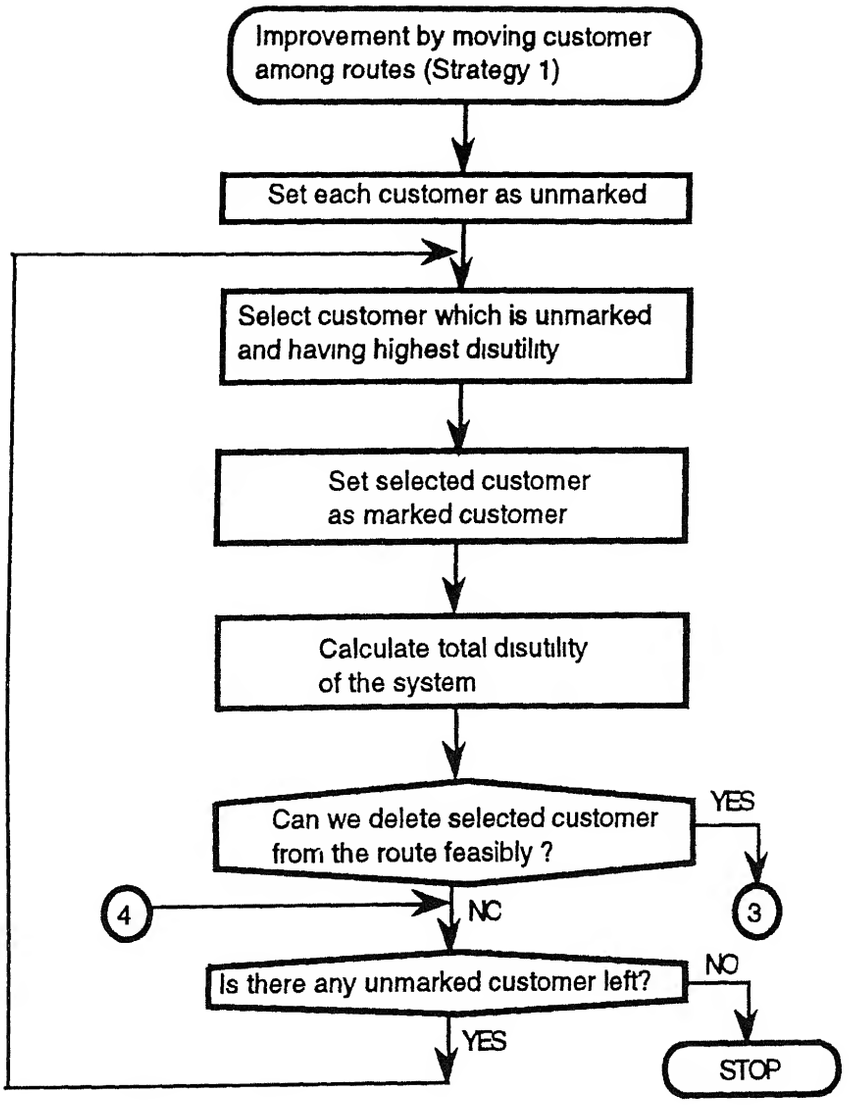

Improvement by moving customer among routes (Strategy 1)
Set each customer as unmarked
Select customer which is unmarked and having highest disutility
Set selected customer as marked customer
Calculate total disutility of the system
Can we delete selected customer from the route feasibly ?
YES
NO
3
4
Is there any unmarked customer left?
NO
YES
STOP

Figure A.6

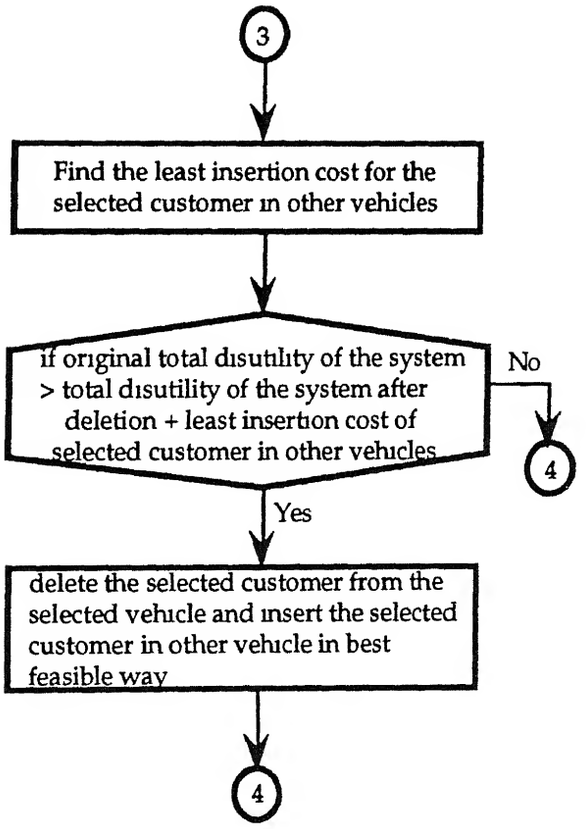
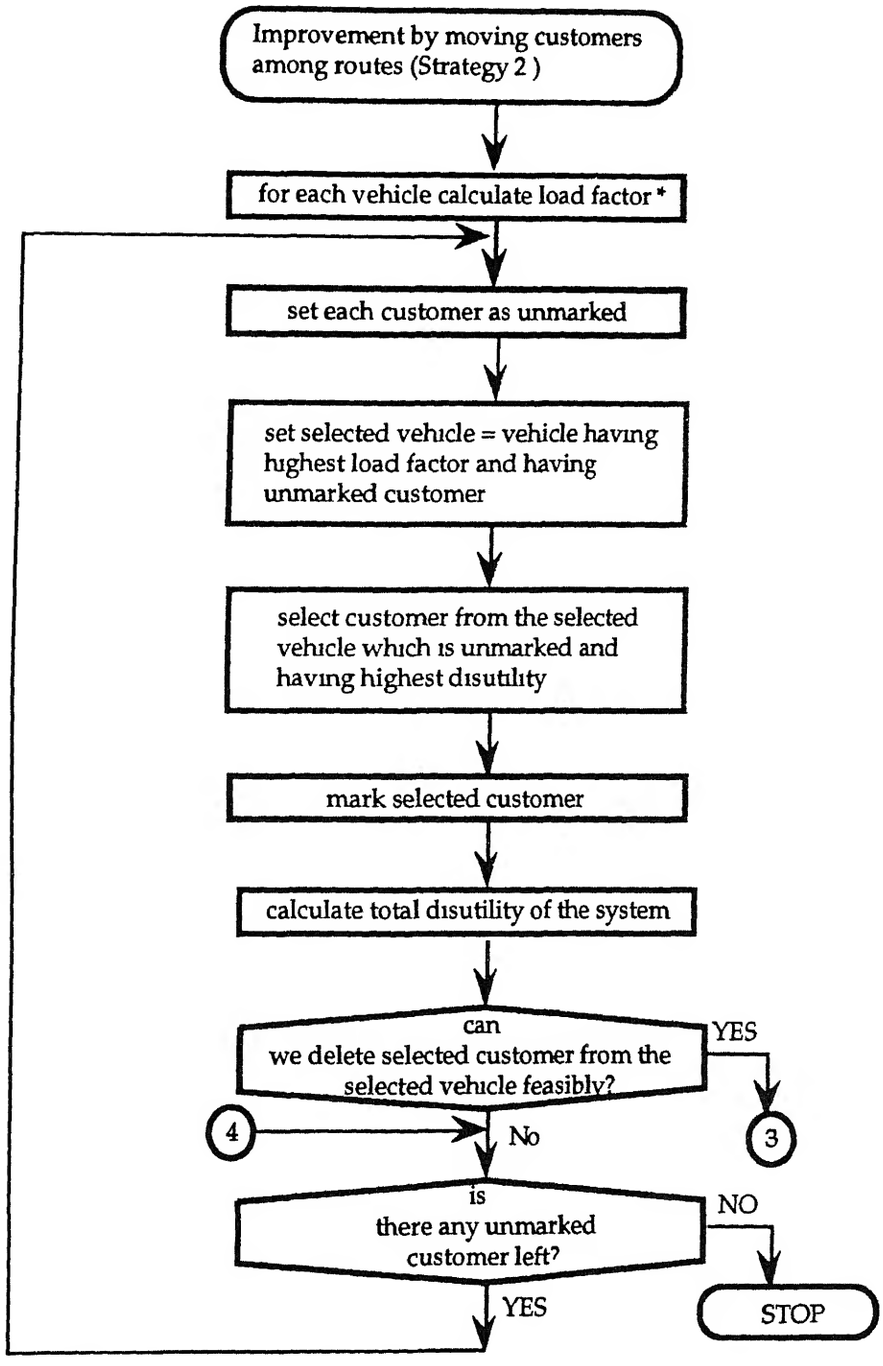

3
Find the least insertion cost for the selected customer in other vehicles
if original total disutility of the system > total disutility of the system after deletion + least insertion cost of selected customer in other vehicles
No
4
Yes
delete the selected customer from the selected vehicle and insert the selected customer in other vehicle in best feasible way
4

Figure A.7



Flow chart A.8

* Any system parameter can be choosen as load factor of a vehicle. We have choosen the service time as the load factor of the vehicle.

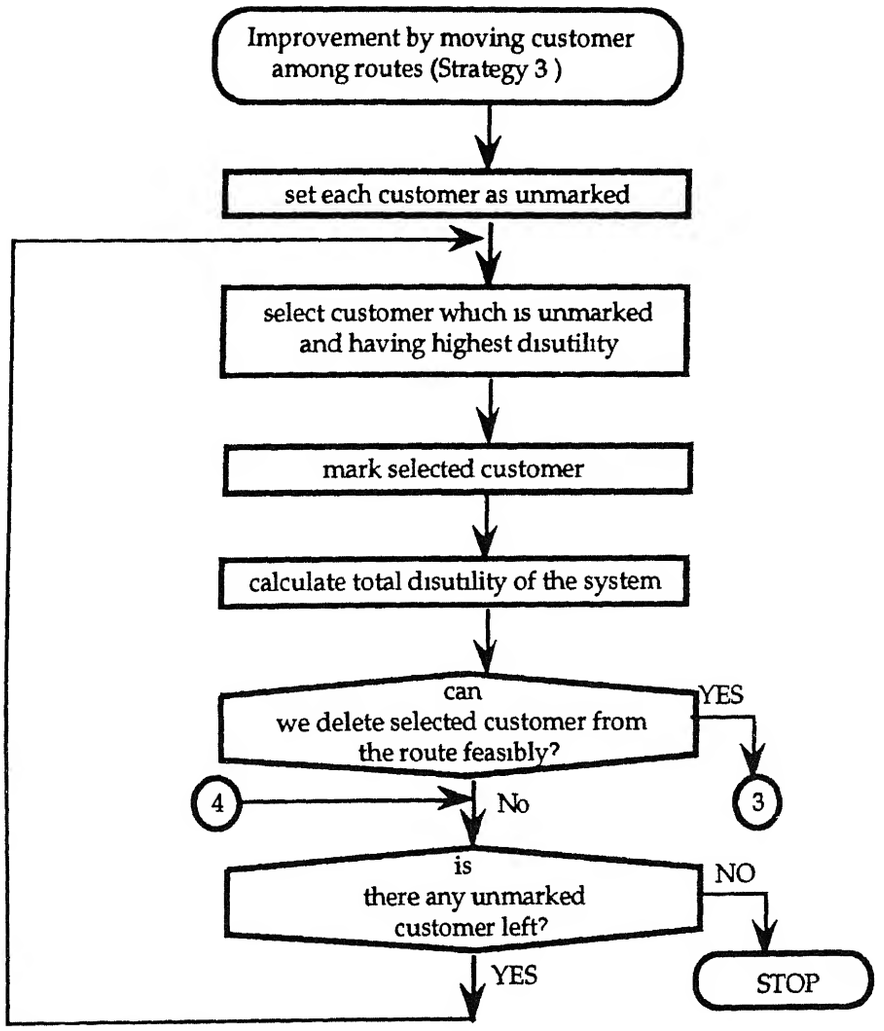

Improvement by moving customer among routes (Strategy 3)
set each customer as unmarked
select customer which is unmarked and having highest disutility
mark selected customer
calculate total disutility of the system
can we delete selected customer from the route feasibly?
YES
No
4
3
is there any unmarked customer left?
NO
YES
STOP

Figure A.9

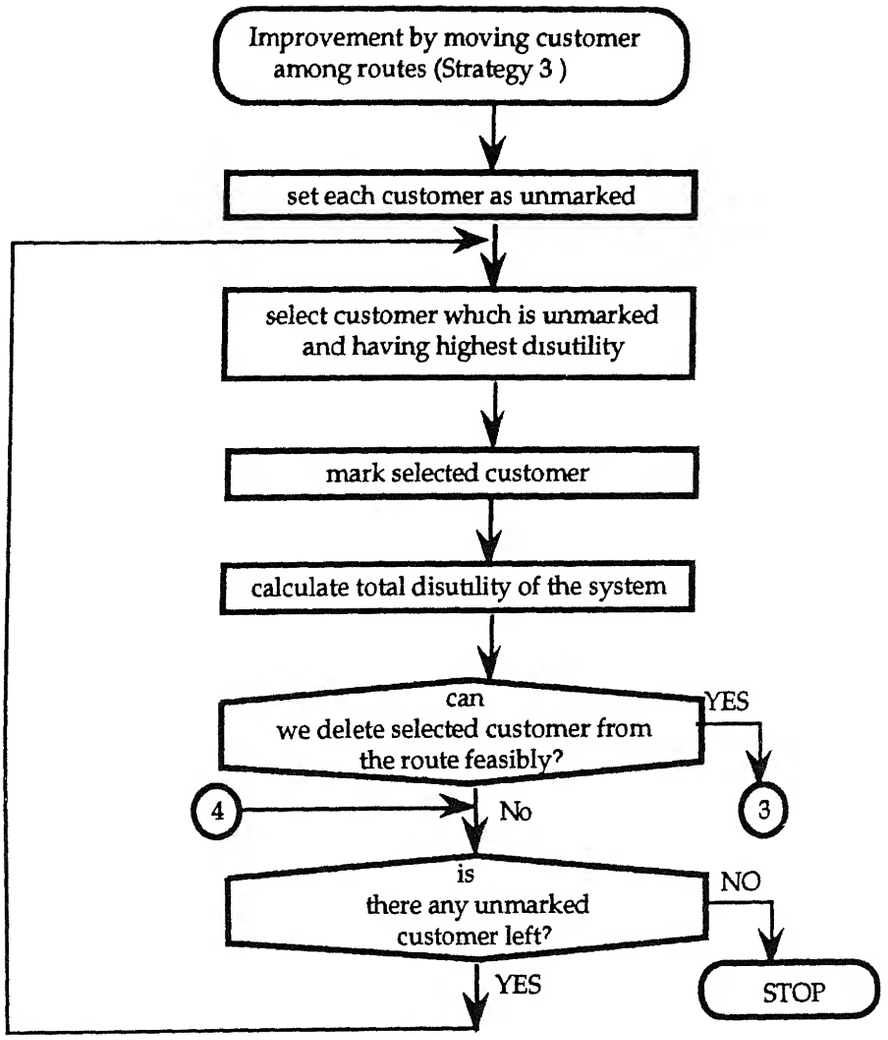

Improvement by moving customer among routes (Strategy 3)
set each customer as unmarked
select customer which is unmarked and having highest disutility
mark selected customer
calculate total disutility of the system
can we delete selected customer from the route feasibly?
YES
3
4
No
is there any unmarked customer left?
NO
STOP
YES

Figure A.9

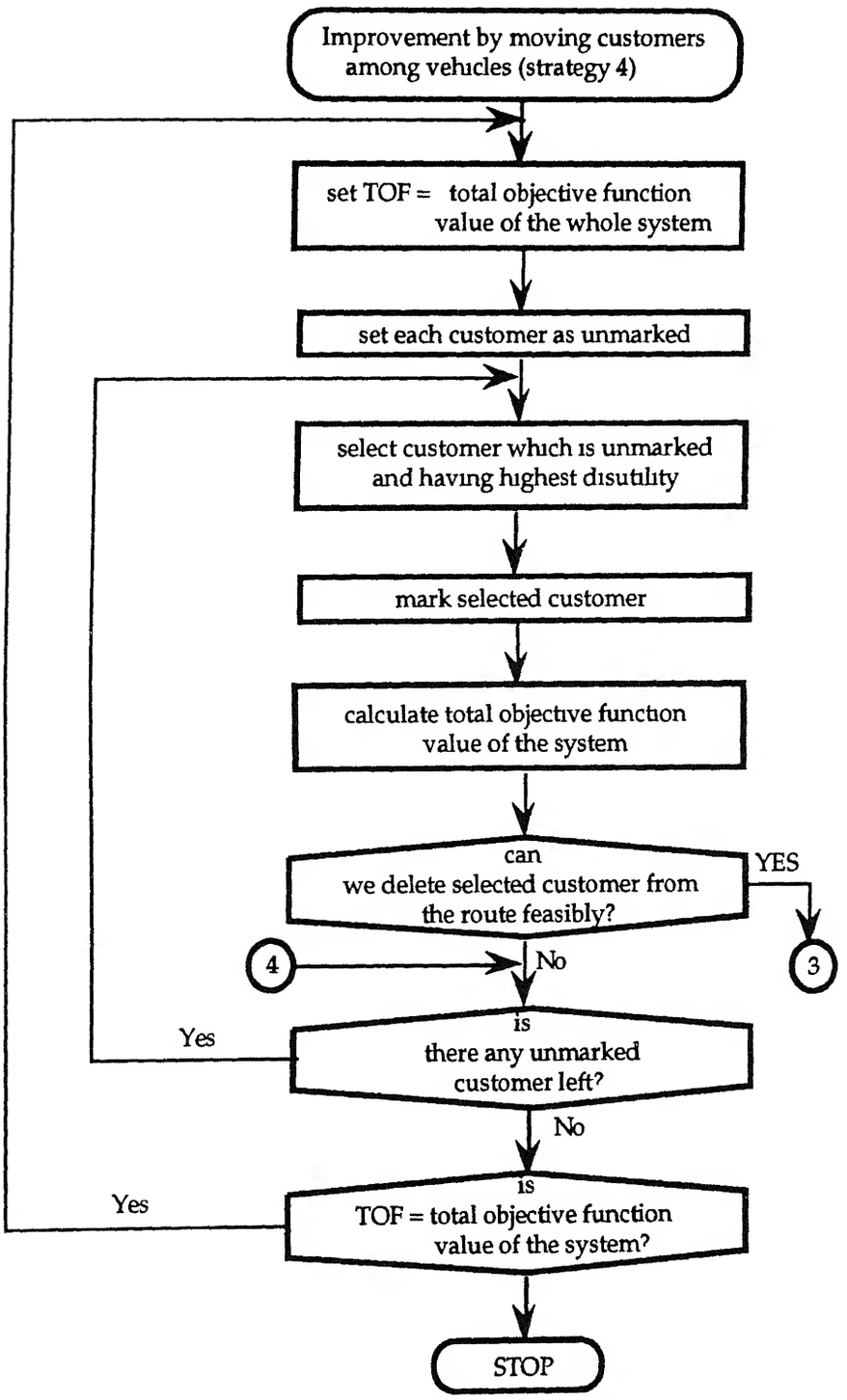

Improvement by moving customers among vehicles (strategy 4)
set TOF = total objective function value of the whole system
set each customer as unmarked
select customer which is unmarked and having highest disutility
mark selected customer
calculate total objective function value of the system
can we delete selected customer from the route feasibly?
YES
4
No
3
is there any unmarked customer left?
Yes
No
is TOF = total objective function value of the system?
Yes
STOP

Figure A.10

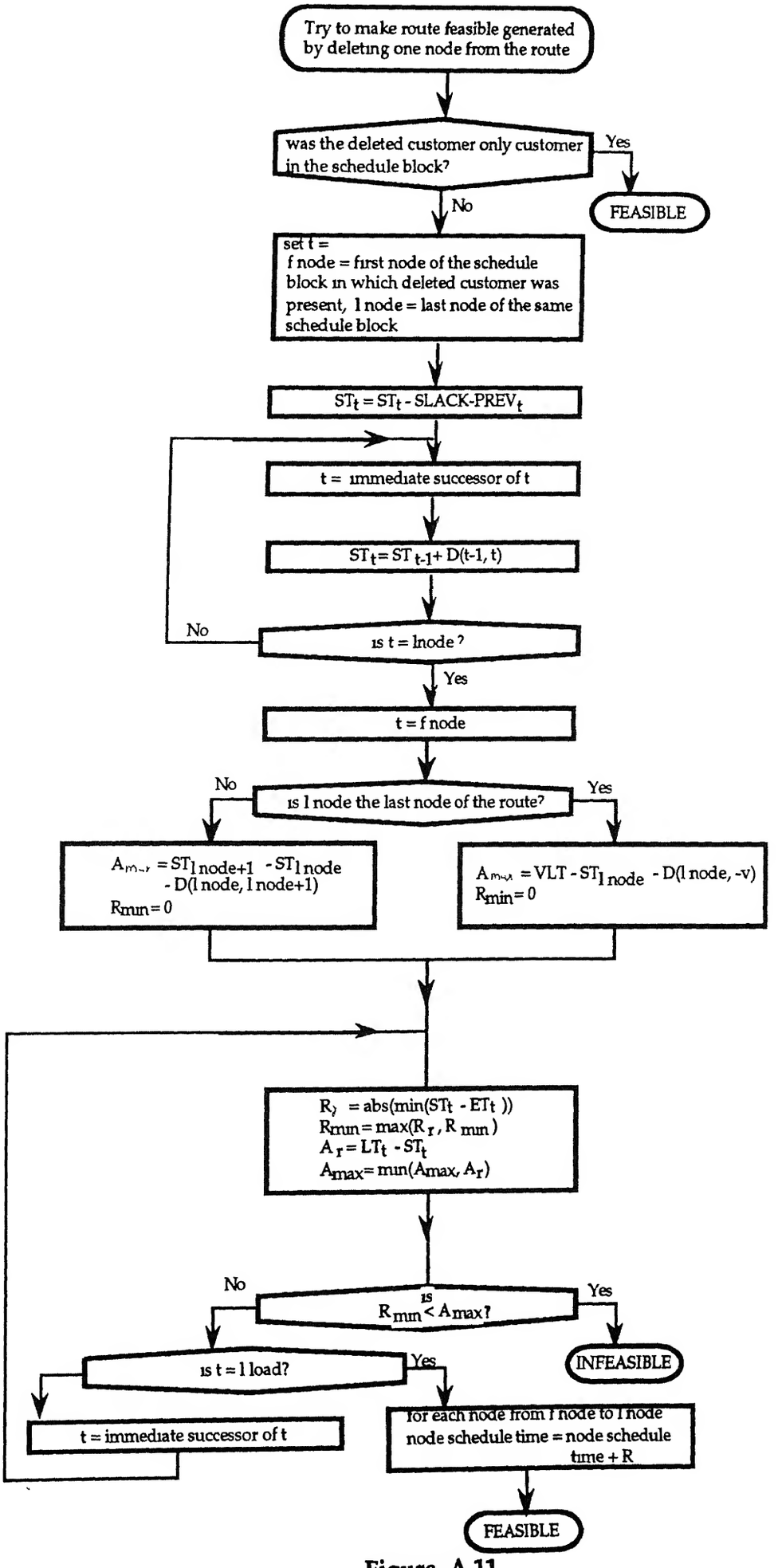

Try to make route feasible generated by deleting one node from the route
was the deleted customer only customer in the schedule block?
Yes
FEASIBLE
No
set t = f node = first node of the schedule block in which deleted customer was present, l node = last node of the same schedule block
$ST_t = ST_t$ - SLACK-PREV$_t$
t = immediate successor of t
$ST_t = ST_{t-1} + D(t-1, t)$
No
is t = lnode ?
Yes
t = f node
No
is l node the last node of the route?
Yes
$A_{max} = ST_{l\,node+1} - ST_{l\,node} - D(l\,node, l\,node+1)$
$R_{min} = 0$
$A_{max} = VLT - ST_{l\,node} - D(l\,node, -v)$
$R_{min} = 0$
$R_t = abs(min(ST_t - ET_t))$
$R_{min} = max(R_t, R_{min})$
$A_t = LT_t - ST_t$
$A_{max} = min(A_{max}, A_t)$
No
is $R_{min} < A_{max}$?
Yes
INFEASIBLE
is t = l load?
Yes
t = immediate successor of t
for each node from f node to l node node schedule time = node schedule time + R
FEASIBLE

Figure A.11

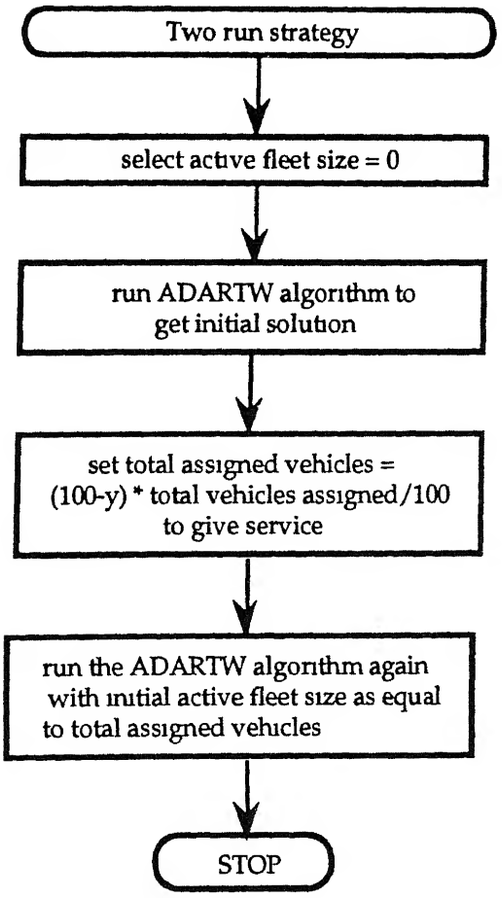

Two run strategy
select active fleet size = 0
run ADARTW algorithm to get initial solution
set total assigned vehicles = (100-y) * total vehicles assigned/100 to give service
run the ADARTW algorithm again with initial active fleet size as equal to total assigned vehicles
STOP

Figure A.12

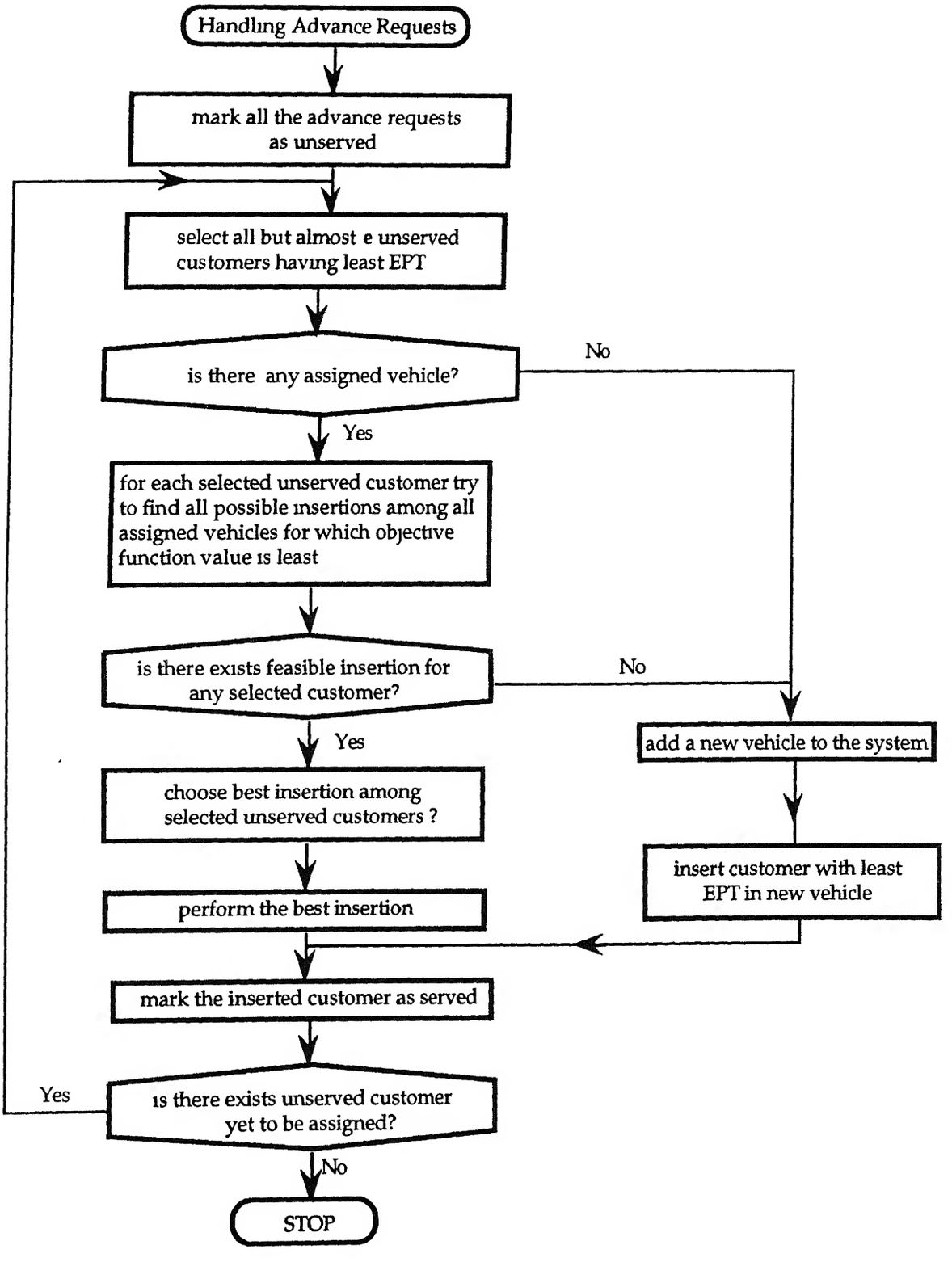

Handling Advance Requests
mark all the advance requests as unserved
select all but almost e unserved customers having least EPT
is there any assigned vehicle?
No
Yes
for each selected unserved customer try to find all possible insertions among all assigned vehicles for which objective function value is least
is there exists feasible insertion for any selected customer?
No
Yes
add a new vehicle to the system
choose best insertion among selected unserved customers ?
insert customer with least EPT in new vehicle
perform the best insertion
mark the inserted customer as served
Yes
is there exists unserved customer yet to be assigned?
No
STOP

Figure A.13

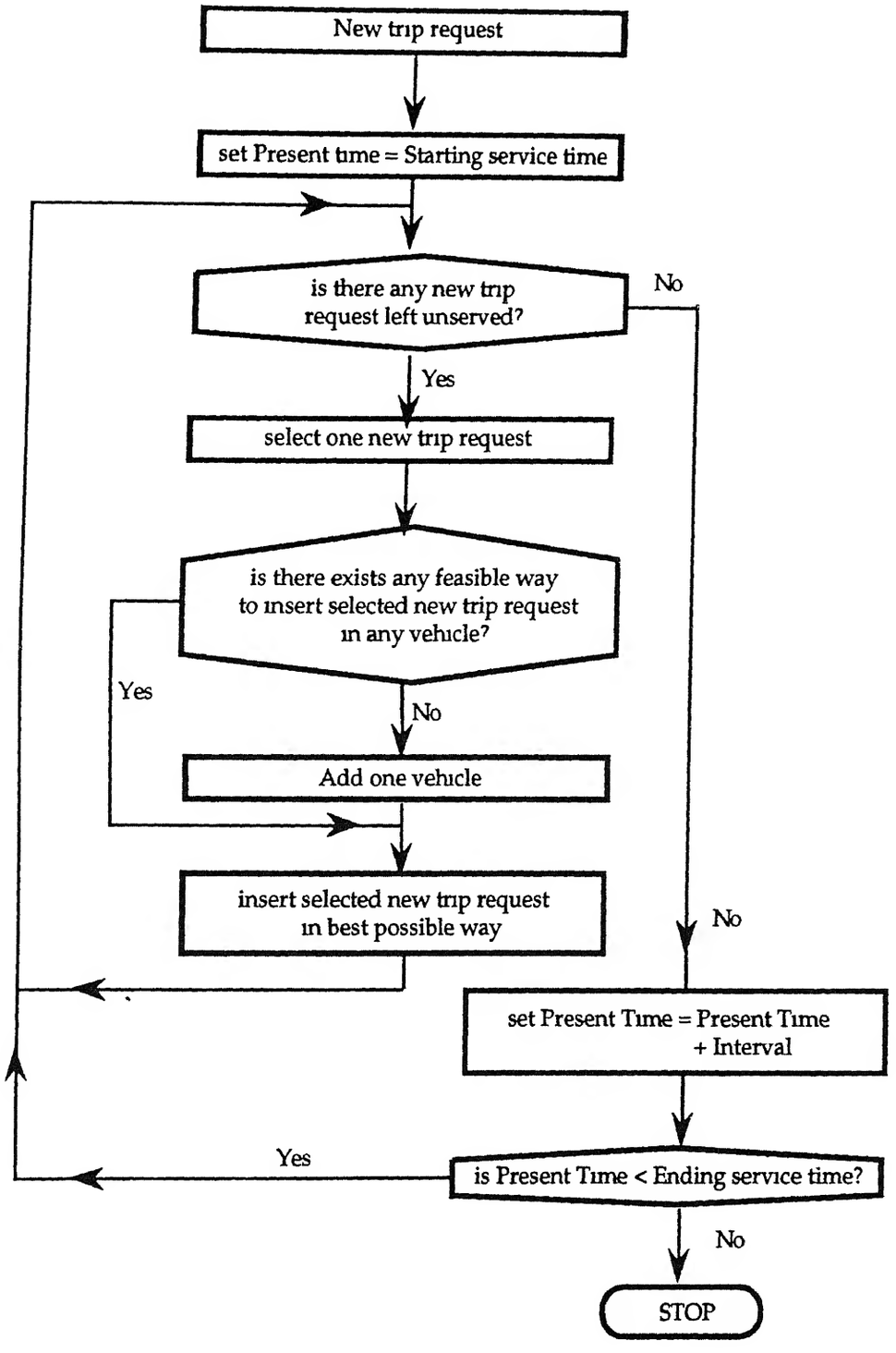

New trip request
set Present time = Starting service time
is there any new trip request left unserved?
No
Yes
select one new trip request
is there exists any feasible way to insert selected new trip request in any vehicle?
Yes
No
Add one vehicle
insert selected new trip request in best possible way
No
set Present Time = Present Time + Interval
Yes
is Present Time < Ending service time?
No
STOP

Figure A.14

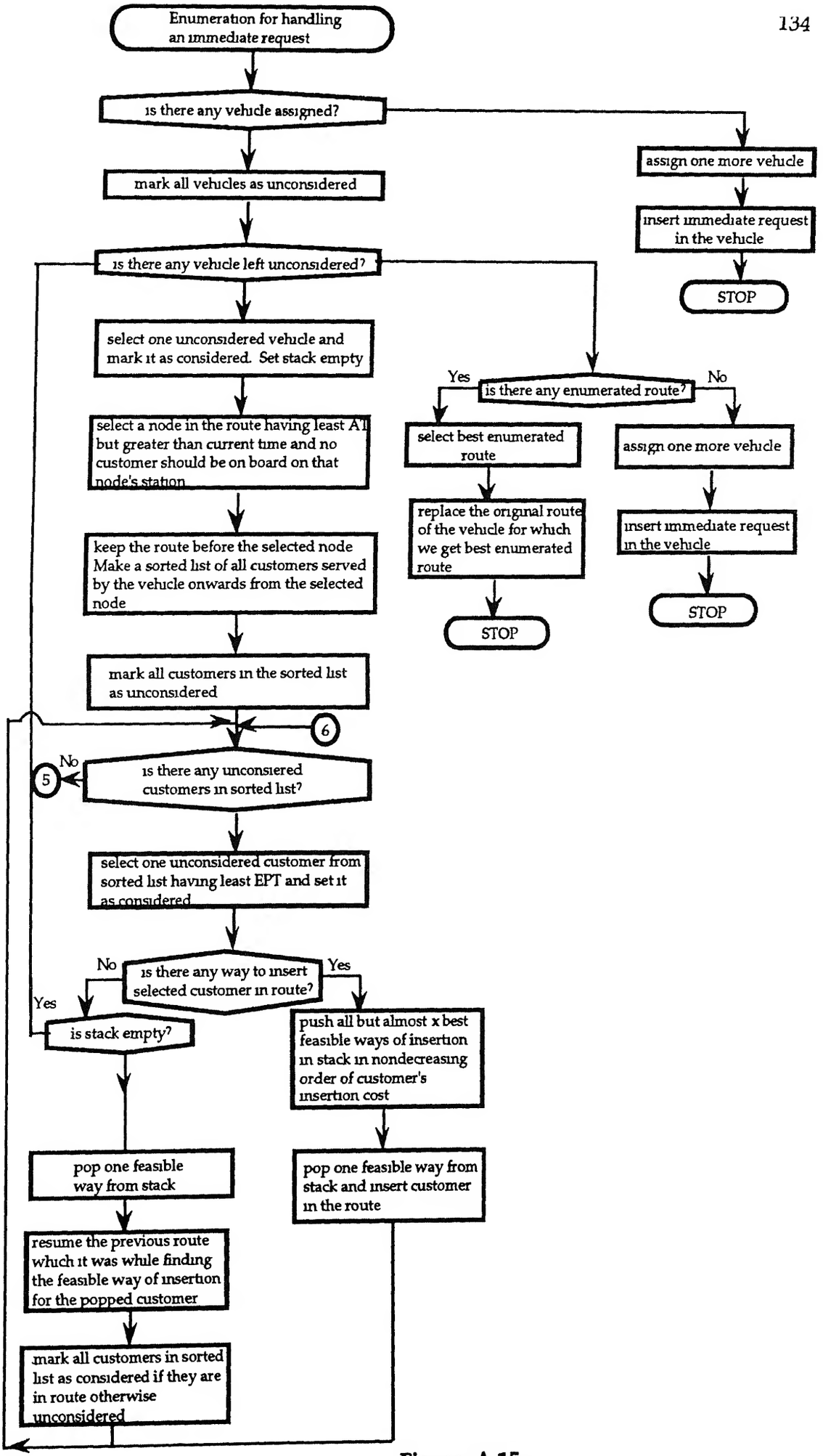

Enumeration for handling an immediate request
is there any vehicle assigned?
assign one more vehicle
insert immediate request in the vehicle
STOP
mark all vehicles as unconsidered
is there any vehicle left unconsidered?
select one unconsidered vehicle and mark it as considered. Set stack empty
Yes
is there any enumerated route?
No
select best enumerated route
assign one more vehicle
select a node in the route having least AT but greater than current time and no customer should be on board on that node's station
replace the original route of the vehicle for which we get best enumerated route
insert immediate request in the vehicle
STOP
STOP
keep the route before the selected node Make a sorted list of all customers served by the vehicle onwards from the selected node
mark all customers in the sorted list as unconsidered
6
No
5
is there any unconsiered customers in sorted list?
select one unconsidered customer from sorted list having least EPT and set it as considered
No
is there any way to insert selected customer in route?
Yes
Yes
is stack empty?
push all but almost x best feasible ways of insertion in stack in nondecreasing order of customer's insertion cost
pop one feasible way from stack
pop one feasible way from stack and insert customer in the route
resume the previous route which it was while finding the feasible way of insertion for the popped customer
mark all customers in sorted list as considered if they are in route otherwise unconsidered

Figure A.15

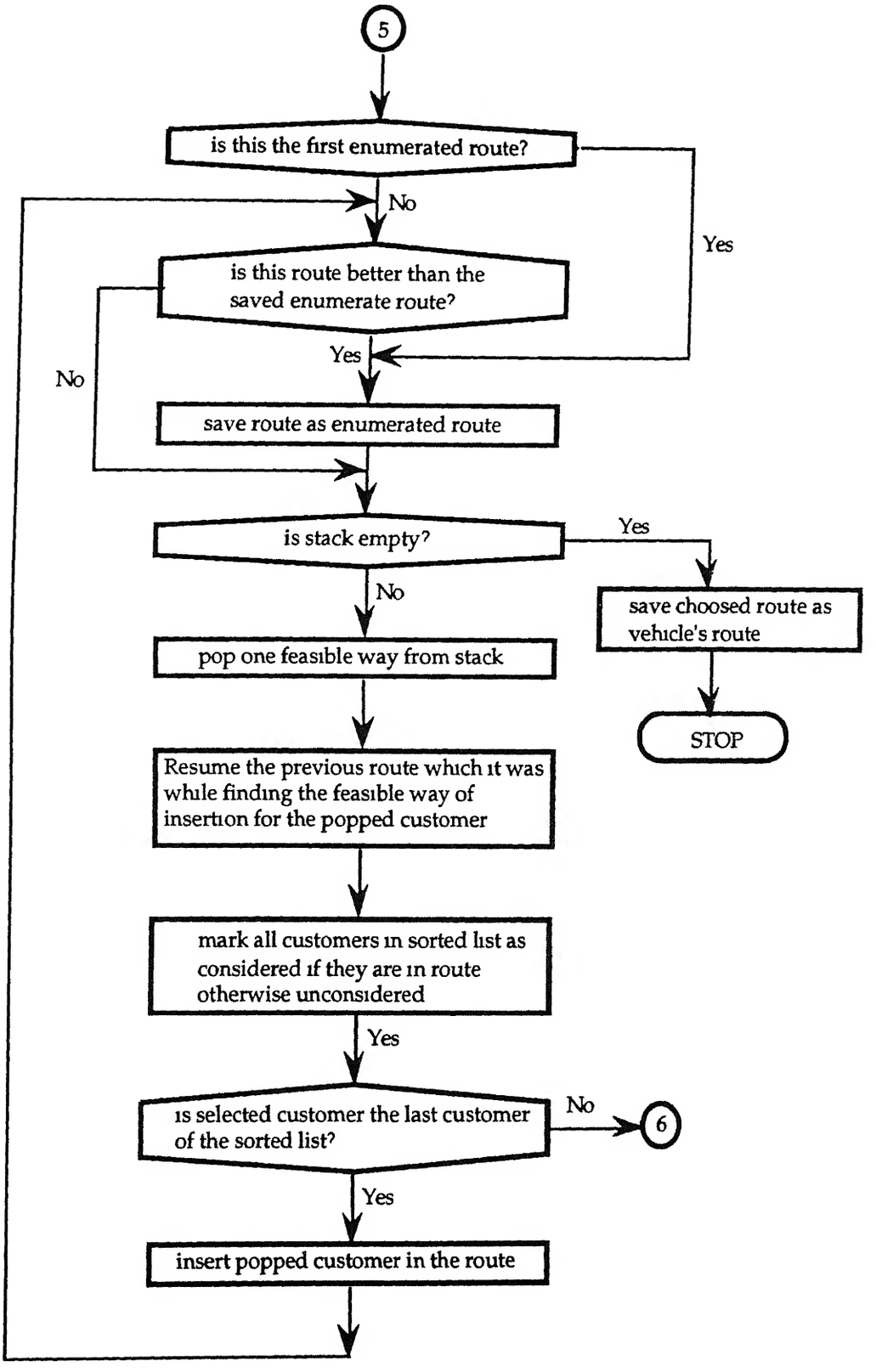

5
is this the first enumerated route?
Yes
No
is this route better than the saved enumerate route?
No
Yes
save route as enumerated route
is stack empty?
Yes
No
save choosed route as vehicle's route
STOP
pop one feasible way from stack
Resume the previous route which it was while finding the feasible way of insertion for the popped customer
mark all customers in sorted list as considered if they are in route otherwise unconsidered
Yes
is selected customer the last customer of the sorted list?
No
6
Yes
insert popped customer in the route

Figure A.16

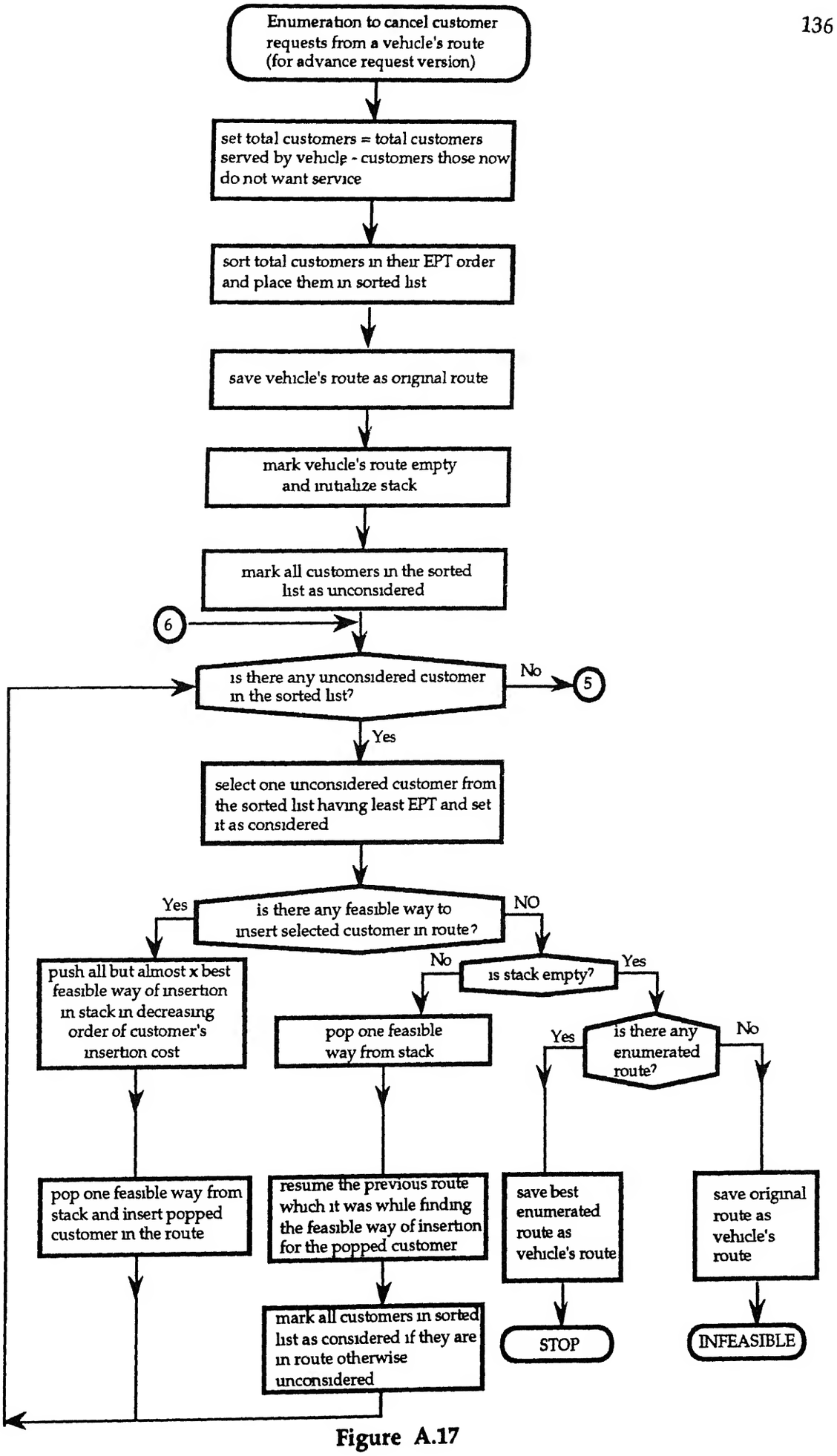

Enumeration to cancel customer requests from a vehicle's route (for advance request version)
set total customers = total customers served by vehicle - customers those now do not want service
sort total customers in their EPT order and place them in sorted list
save vehicle's route as original route
mark vehicle's route empty and initialize stack
mark all customers in the sorted list as unconsidered
6
is there any unconsidered customer in the sorted list?
No
5
Yes
select one unconsidered customer from the sorted list having least EPT and set it as considered
Yes
is there any feasible way to insert selected customer in route?
NO
push all but almost x best feasible way of insertion in stack in decreasing order of customer's insertion cost
No
is stack empty?
Yes
pop one feasible way from stack
Yes
is there any enumerated route?
No
pop one feasible way from stack and insert popped customer in the route
resume the previous route which it was while finding the feasible way of insertion for the popped customer
save best enumerated route as vehicle's route
save original route as vehicle's route
mark all customers in sorted list as considered if they are in route otherwise unconsidered
STOP
INFEASIBLE

Figure A.17

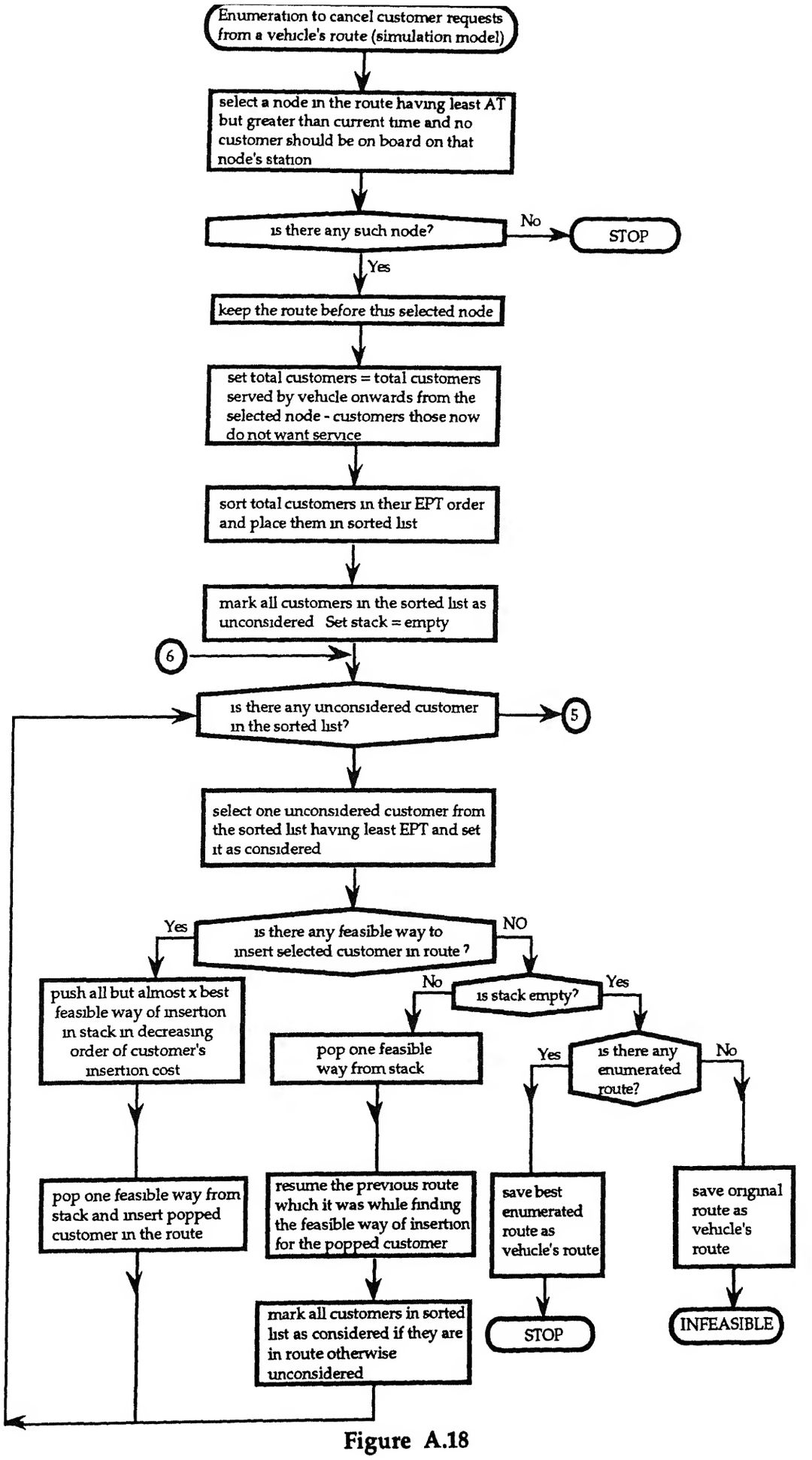

Enumeration to cancel customer requests from a vehicle's route (simulation model)
select a node in the route having least AT but greater than current time and no customer should be on board on that node's station
is there any such node?
No
STOP
Yes
keep the route before this selected node
set total customers = total customers served by vehicle onwards from the selected node - customers those now do not want service
sort total customers in their EPT order and place them in sorted list
mark all customers in the sorted list as unconsidered Set stack = empty
6
is there any unconsidered customer in the sorted list?
5
select one unconsidered customer from the sorted list having least EPT and set it as considered
Yes
is there any feasible way to insert selected customer in route ?
NO
push all but almost x best feasible way of insertion in stack in decreasing order of customer's insertion cost
No
is stack empty?
Yes
pop one feasible way from stack
Yes
is there any enumerated route?
No
pop one feasible way from stack and insert popped customer in the route
resume the previous route which it was while finding the feasible way of insertion for the popped customer
save best enumerated route as vehicle's route
save original route as vehicle's route
mark all customers in sorted list as considered if they are in route otherwise unconsidered
STOP
INFEASIBLE

Figure A.18

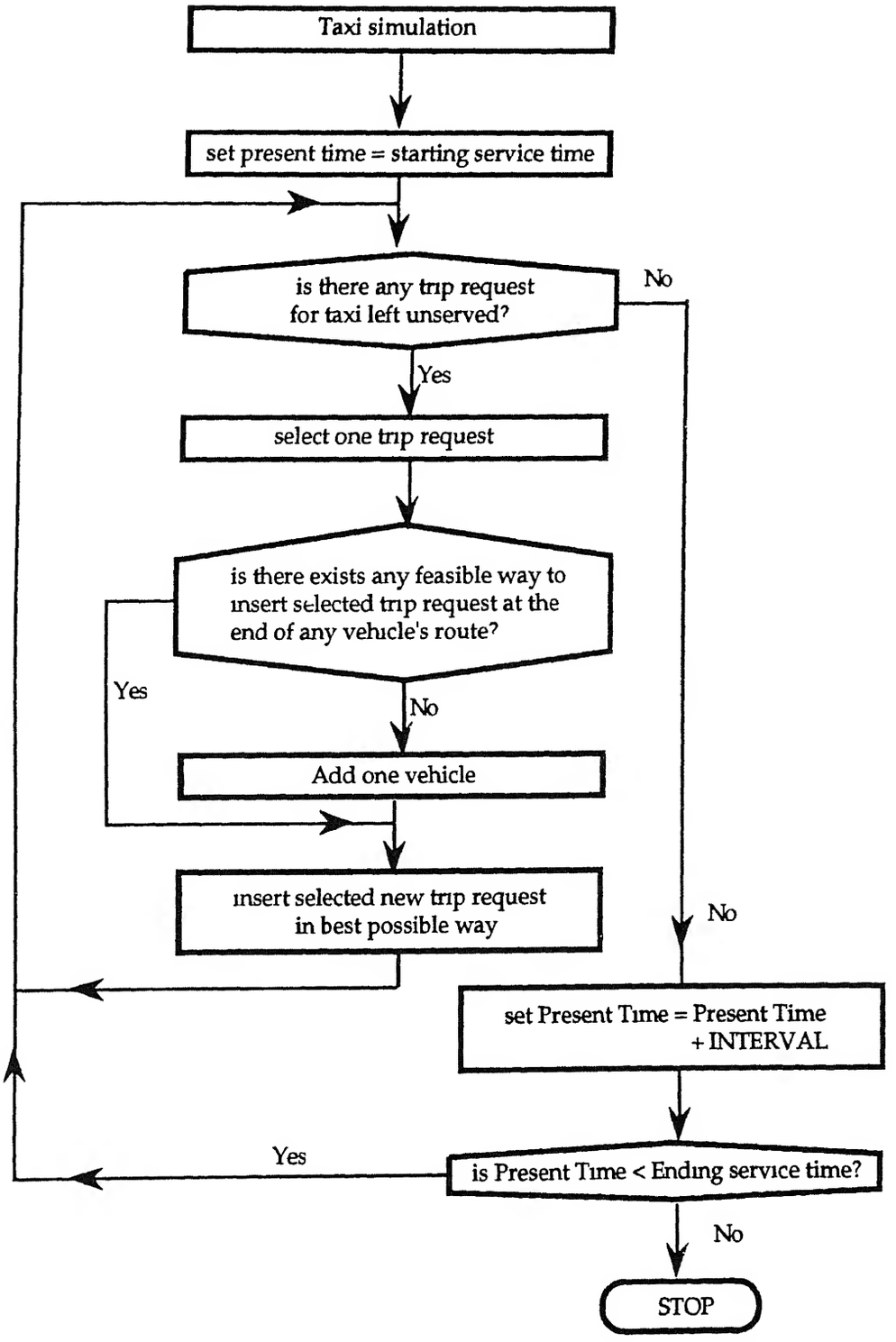

Taxi simulation
set present time = starting service time
is there any trip request for taxi left unserved?
No
Yes
select one trip request
is there exists any feasible way to insert selected trip request at the end of any vehicle's route?
Yes
No
Add one vehicle
insert selected new trip request in best possible way
No
set Present Time = Present Time + INTERVAL
Yes
is Present Time < Ending service time?
No
STOP

Figure A.19